CS Space
Клуб технологий и науки

Вычисления на видеокартах

Vulkan
OpenCL™
NVIDIA
CUDA

Лекция 3

- Сложение матриц и анализ скорости
- Напоминание о ментальной модели
- Локальная память и барьеры
- Суммирование чисел в массиве
- Новые задания

polarnick239@gmail.com
Николай Полярный

Глава 1: сложение матриц и анализ скорости

Speed of Light анализ, профилирование, Structure of Arrays и Array of Structures

Домашнее задание: суммирование матриц

```
Using device #1: API: CUDA+OpenCL+Vulkan. GPU. NVIDIA GeForce RTX 4090 (CUDA 12090).
Using CUDA API...
matrices size: 16384x8192 = 3 * 512 MB

Running BAD matrix kernel...
a + b matrix kernel median VRAM bandwidth: 198.66 GB/s
```

Могу ли я дальше это ускорить? Какой мой следующий шаг?

Домашнее задание: суммирование матриц

```
Using device #1: API: CUDA+OpenCL+Vulkan. GPU. NVIDIA GeForce RTX 4090 (CUDA 12090).
Using CUDA API...
matrices size: 16384x8192 = 3 * 512 MB

Running BAD matrix kernel...
a + b matrix kernel median VRAM bandwidth: 198.66 GB/s

Running GOOD matrix kernel...
a + b matrix kernel median VRAM bandwidth: 833.287 GB/s
```

Могу ли я дальше это ускорить? Какой мой следующий шаг?

Домашнее задание: суммирование матриц

```
Using device #1: API: CUDA+OpenCL+Vulkan. GPU. NVIDIA GeForce RTX 4090 (CUDA 12090).
Using CUDA API...
matrices size: 16384x8192 = 3 * 512 MB

Running BAD matrix kernel...
a + b matrix kernel median VRAM bandwidth: 198.66 GB/s

Running GOOD matrix kernel...
a + b matrix kernel median VRAM bandwidth: 833.287 GB/s
```

Могу ли я дальше это ускорить? Какой мой следующий шаг?

1) **Speed-of-Light** (**SoL**) анализ - позволяет быстро принимать решения, встраивать Live-профилирование в продукт навсегда

Могу ли я ускорить с помощью оптимизаций вычислений?


Все зависит от узкого места!

Домашнее задание: суммирование матриц

```
Using device #1: API: CUDA+OpenCL+Vulkan. GPU. NVIDIA GeForce RTX 4090 (CUDA 12090).
Using CUDA API...
matrices size: 16384x8192 = 3 * 512 MB

Running BAD matrix kernel...
a + b matrix kernel median VRAM bandwidth: 198.66 GB/s

Running GOOD matrix kernel...
a + b matrix kernel median VRAM bandwidth: 833.287 GB/s
```

Model	Memory			Processing power (TFLOPS)			
	Size (GB)	Band-width (GB/s)	Bus width (bit)	Half (boost)	Single (boost)	Double (boost)	Tensor compute (sparse)
GeForce RTX 4090[74][75]	24	1008	384	73.07 (82.58)	73.07 (82.58)	1.142 (1.290)	330.3 [660.6]

Могу ли я дальше это ускорить? Какой мой следующий шаг?

1) **Speed-of-Light** (**SoL**) анализ - позволяет быстро принимать решения, встраивать Live-профилирование в продукт навсегда

Домашнее задание: суммирование матриц

```
Using device #1: API: CUDA+OpenCL+Vulkan. GPU. NVIDIA GeForce RTX 4090 (CUDA 12090).
Using CUDA API...
matrices size: 16384x8192 = 3 * 512 MB

Running BAD matrix kernel...
a + b matrix kernel median VRAM bandwidth: 198.66 GB/s

Running GOOD matrix kernel...
a + b matrix kernel median VRAM bandwidth: 833.287 GB/s
```

Могу ли я дальше это ускорить? Какой мой следующий шаг?

1) **Speed-of-Light** (**SoL**) анализ - позволяет быстро принимать решения, встраивать Live-профилирование в продукт навсегда
2) **Профилировщик** - не требует аналитики, показывает как есть

NVIDIA Nsight™ Compute

Профилировщик

```
Running BAD matrix kernel...
a + b matrix kernel median VRAM bandwidth: 198.66 GB/s
```

ID	Estimated Speedup [%]	Function Name	Demangled Name	Duration [ms] (25,42 ms)	Runtime Improvement [ms] (8,01 ms)	Compute Throughput [%]	Memo
0	87.45	aplusb_matrix_bad	aplusb_matrix_bad..	7,84	6,85	1,88	
1	5.24	aplusb_matrix_good	aplusb_matrix_goo...	1,70	0,09	8,65	
2	5.11	aplusb_matrix_good	aplusb_matrix_goo...	1,69	0,09	8,71	

The following performance optimization opportunities were discovered for this result. Follow the rule links to see more context on the Details page.
Note: *Speedup estimates provide upper bounds for the optimization potential of a kernel assuming its overall algorithmic structure is kept unchanged.*

Uncoalesced Global Accesses
Est. Speedup: 87.45%
This kernel has uncoalesced global accesses resulting in a total of 352321536 excessive sectors (88% of the total 402653184 sectors). Check the L2 Theoretical Sectors Global Excessive table for the primary source locations. The CUDA Programming Guide has additional information on reducing uncoalesced device memory accesses.

L2 Global Load Access Pattern
Est. Speedup: 77.74%
The memory access pattern for global loads from L2 might not be optimal. On average, only 4.0 of the 32 bytes transmitted per sector are utilized by each thread. This applies to the 99.9% of sectors missed in L1TEX. This could possibly be caused by a stride between threads. Check the Source Counters section for uncoalesced global loads.

L2 Global Store Access Pattern
Est. Speedup: 77.07%
The memory access pattern for global stores to L2 might not be optimal. On average, only 4.0 of the 32 bytes transmitted per sector are utilized by each thread. This applies to the 99.0% of sectors missed in L1TEX. This could possibly be caused by a stride between threads. Check the Source Counters section for uncoalesced global stores.

NVIDIA Nsight™ Compute

Профилировщик

```
Running GOOD matrix kernel...
a + b matrix kernel median VRAM bandwidth: 833.287 GB/s
```

ID	Estimated Speedup [%]	Function Name	Demangled Name	Duration [ms] (25,42 ms)	Runtime Improvement [ms] (8,01 ms)	Compute Throughput [%]	Memo
0	87.45	aplusb_matrix_bad	aplusb_matrix_bad..	7,84	6,85	1,88	
1	5.24	aplusb_matrix_good	aplusb_matrix_go...	1,70	0,09	8,65	
2	5.11	aplusb_matrix_good	aplusb_matrix_goo...	1,69	0,09	8,71	

The following performance optimization opportunities were discovered for this result. Follow the rule links to see more context on the Details page.
Note: *Speedup estimates provide upper bounds for the optimization potential of a kernel assuming its overall algorithmic structure is kept unchanged.*

Achieved Occupancy
Est. Speedup: 5.24%

The difference between calculated theoretical (100.0%) and measured achieved occupancy (85.1%) can be the result of warp scheduling overheads or workload imbalances during the kernel execution. Load imbalances can occur between warps within a block as well as across blocks of the same kernel. See the CUDA Best Practices Guide for more details on optimizing occupancy.

Long Scoreboard Stalls
Est. Speedup: 5.24%

On average, each warp of this workload spends 215.6 cycles being stalled waiting for a scoreboard dependency on a L1TEX (local, global, surface, texture) operation. Find the instruction producing the data being waited upon to identify the culprit. To reduce the number of cycles waiting on L1TEX data accesses verify the memory access patterns are optimal for the target architecture, attempt to increase cache hit rates by increasing data locality (coalescing), or by changing the cache configuration. Consider moving frequently used data to shared memory. This stall type represents about 96.2% of the total average of 224.2 cycles between issuing two instructions.

NVIDIA Nsight™ Compute

Профилировщик

```
Running GOOD matrix kernel...
a + b matrix kernel median VRAM bandwidth: 833.287 GB/s
```

ID	Estimated Speedup [%]	Function Name	Demangled Name	Duration [ms] (25,42 ms)	Runtime Improvement [ms] (8,01 ms)	Compute Throughput [%]	Memo
0	87.45	aplusb_matrix_bad	aplusb_matrix_bad..	7,84	6,85	1,88	
1	5.24	aplusb_matrix_good	aplusb_matrix_go...	1,70	0,09	8,65	
2	5.11	aplusb_matrix_good	aplusb_matrix_goo...	1,69	0,09	8,71	

The following performance optimization opportunities were discovered for this result. Follow the rule links to see more context on the Details page.
Note: *Speedup estimates provide upper bounds for the optimization potential of a kernel assuming its overall algorithmic structure is kept unchanged.*

Achieved Occupancy
Est. Speedup: 5.24%

The difference between calculated theoretical (100.0%) and measured achieved occupancy (85.1%) can be the result of warp scheduling overheads or workload imbalances during the kernel execution. Load imbalances can occur between warps within a block as well as across blocks of the same kernel. See the CUDA Best Practices Guide for more details on optimizing occupancy.

Long Scoreboard Stalls
Est. Speedup: 5.24%

On average, each warp of this workload spends 215.6 cycles being stalled waiting for a scoreboard dependency on a L1TEX (local, global, surface, texture) operation. Find the instruction producing the data being waited upon to identify the culprit. To reduce the number of cycles waiting on L1TEX data accesses verify the memory access patterns are optimal for the target architecture, attempt to increase cache hit rates by increasing data locality (coalescing), or by changing the cache configuration. Consider moving frequently used data to shared memory. This stall type represents about 96.2% of the total average of 224.2 cycles between issuing two instructions.

Array of Structures (AoS) vs Structure of Arrays (SoA)

```
class Point4f {
    float x;
    float y;
    float z;
    float w;
};
```

Array of Structures (AoS) vs Structure of Arrays (SoA)

```
class Point4f {
    float x;
    float y;
    float z;
    float w;
};
```

```
__global__ void kernel_process_points(
    const Point4f* points,
    unsigned int   n)
{
    const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;

    float x = points[index].x;
    // ...
```

coalesced memory access pattern?

Array of Structures (AoS) vs **Structure of Arrays (SoA)**

```
__global__ void kernel_process_points(
    const float* points_x,
    const float* points_y,
    const float* points_z,
    const float* points_w,
    unsigned int   n)
{
    const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;

    float x = points_x[index];
    // ...
```

coalesced memory access pattern?


Глава 2: ментальная модель
Лилипуты, клоуны, официанты, граница между API и железом

Мысленная модель, стандарт и API, железо

1) **Лилипут** = **???**

Мысленная модель, стандарт и API, железо

1) **Лилипут** = поток с его личными:
   1.1) **???**

Мысленная модель, стандарт и API, железо

1) **Лилипут** = поток с его личными:
   1.1) значениями **переменных** (промежуточные вычисления, **регистры**)
   1.2) номером **workItem** (номер задачи)
   1.3) с его принадлежностью **workGroup** (а на уровне железа - **warp** и **SM**)

Что из **этого всего** - мысленная модель, стандарт и API, железо???

Мысленная модель, стандарт и API, железо

1) **Лилипут** = поток с его личными:
   1.1) значениями **переменных** (промежуточные вычисления, **регистры**)
   1.2) номером **workItem** (номер задачи)
   1.3) с его принадлежностью **workGroup** (а на уровне железа - **warp** и **SM**)

Мысленная модель, стандарт и API, железо

1) **Лилипут** = поток с его личными:
   1.1) значениями **переменных** (промежуточные вычисления, **регистры**)
   1.2) номером **workItem** (номер задачи)
   1.3) с его принадлежностью **workGroup** (а на уровне железа - **warp** и **SM**)
2) **Машинка с 32 лилипутами** = **???**

Мысленная модель, стандарт и API, железо

1) **Лилипут** = поток с его личными:
   1.1) значениями **переменных** (промежуточные вычисления, **регистры**)
   1.2) номером **workItem** (номер задачи)
   1.3) с его принадлежностью **workGroup** (а на уровне железа - **warp** и **SM**)
2) **Машинка с 32 лилипутами** = **warp:**
   2.1) есть общий **instruction pointer**
   2.2) может быть **active** - все данные подгружены в регистры - либо сейчас считаем у **станка**-**ALU**, либо ждем когда он освободится
   2.3) может быть **inactive** - ждет когда **транзакции** подгрузят все запрошенные **cache lines** из **VRAM**

Что из **этого всего** - мысленная модель, стандарт и API, железо???

Мысленная модель, стандарт и API, железо

1) **Лилипут** = поток с его личными:
   1.1) значениями **переменных** (промежуточные вычисления, **регистры**)
   1.2) номером **workItem** (номер задачи)
   1.3) с его принадлежностью **workGroup** (а на уровне железа - **warp** и **SM**)
2) **Машинка с 32 лилипутами** = **warp:**
   2.1) есть общий **instruction pointer**
   2.2) может быть **active** - все данные подгружены в регистры - либо сейчас считаем у **станка**-**ALU**, либо ждем когда он освободится
   2.3) может быть **inactive** - ждет когда **транзакции** подгрузят все запрошенные **cache lines** из **VRAM**

Мысленная модель, стандарт и API, железо

1) **Лилипут** = поток с его личными:
   1.1) значениями **переменных** (промежуточные вычисления, **регистры**)
   1.2) номером **workItem** (номер задачи)
   1.3) с его принадлежностью **workGroup** (а на уровне железа - **warp** и **SM**)
2) **Машинка с 32 лилипутами** = **warp:**
   2.1) есть общий **instruction pointer**
   2.2) может быть **active** - все данные подгружены в регистры - либо сейчас считаем у **станка**-**ALU**, либо ждем когда он освободится
   2.3) может быть **inactive** - ждет когда **транзакции** подгрузят все запрошенные **cache lines** из **VRAM**
3) **Официант** = **???**

Мысленная модель, стандарт и API, железо

1) **Лилипут** = поток с его личными:
   1.1) значениями **переменных** (промежуточные вычисления, **регистры**)
   1.2) номером **workItem** (номер задачи)
   1.3) с его принадлежностью **workGroup** (а на уровне железа - **warp** и **SM**)
2) **Машинка с 32 лилипутами** = **warp:**
   2.1) есть общий **instruction pointer**
   2.2) может быть **active** - все данные подгружены в регистры - либо сейчас считаем у **станка**-**ALU**, либо ждем когда он освободится
   2.3) может быть **inactive** - ждет когда **транзакции** подгрузят все запрошенные **cache lines** из **VRAM**
3) **Официант** = шина коммуникации с **VRAM**:
   3.1) принимает запросы от **warp**-ов и обслуживает их
   3.2) обладает большой **latency**
   3.3) может одновременно обслуживать много запросов и за счет этого большая общая пропускная способность (общее число приносимых **cache lines**)

Что из **этого всего** - мысленная модель, стандарт и API, железо???

Мысленная модель, стандарт и API, железо

1) **Лилипут** = поток с его личными:
   1.1) значениями **переменных** (промежуточные вычисления, **регистры**)
   1.2) номером **workItem** (номер задачи)
   1.3) с его принадлежностью **workGroup** (а на уровне железа - **warp** и **SM**)
2) **Машинка с 32 лилипутами** = **warp:**
   2.1) есть общий **instruction pointer**
   2.2) может быть **active** - все данные подгружены в регистры - либо сейчас считаем у **станка**-**ALU**, либо ждем когда он освободится
   2.3) может быть **inactive** - ждет когда **транзакции** подгрузят все запрошенные **cache lines** из **VRAM**
3) **Официант** = шина коммуникации с **VRAM**:
   3.1) принимает запросы от **warp**-ов и обслуживает их
   3.2) обладает большой **latency**
   3.3) может одновременно обслуживать много запросов и за счет этого большая общая пропускная способность (общее число приносимых **cache lines**)

Мысленная модель, стандарт и API, железо

1) **Лилипут** = поток с его личными:
   1.1) значениями **переменных** (промежуточные вычисления, **регистры**)
   1.2) номером **workItem** (номер задачи)
   1.3) с его принадлежностью **workGroup** (а на уровне железа - **warp** и **SM**)
2) **Машинка с 32 лилипутами** = **warp:**
   2.1) есть общий **instruction pointer**
   2.2) может быть **active** - все данные подгружены в регистры - либо сейчас считаем у **станка**-**ALU**, либо ждем когда он освободится
   2.3) может быть **inactive** - ждет когда **транзакции** подгрузят все запрошенные **cache lines** из **VRAM**
3) **Официант** = шина коммуникации с **VRAM**:
   3.1) принимает запросы от **warp**-ов и обслуживает их
   3.2) обладает большой **latency**
   3.3) может одновременно обслуживать много запросов и за счет этого большая общая пропускная способность (общее число приносимых **cache lines**)
4) **Станки**-**ALU = FP32 + INT32** счеты

Мысленная модель, стандарт и API, железо

1) **Лилипут** = поток с его личными:
   1.1) значениями **переменных** (промежуточные вычисления, **регистры**)
   1.2) номером **workItem** (номер задачи)
   1.3) с его принадлежностью **workGroup** (а на уровне железа - **warp** и **SM**)
2) **Машинка с 32 лилипутами** = **warp:**
   2.1) есть общий **instruction pointer**
   2.2) может быть **active** - все данные подгружены в регистры - либо сейчас считаем у **станка**-**ALU**, либо ждем когда он освободится
   2.3) может быть **inactive** - ждет когда **транзакции** подгрузят все запрошенные **cache lines** из **VRAM**
3) **Официант** = шина коммуникации с **VRAM**:
   3.1) принимает запросы от **warp**-ов и обслуживает их
   3.2) обладает большой **latency**
   3.3) может одновременно обслуживать много запросов и за счет этого большая общая пропускная способность (общее число приносимых **cache lines**)
4) **Станки**-**ALU = FP32 + INT32** счеты

Их столько же сколько Лилипутов???

Мысленная модель, стандарт и API, железо

1) **Лилипут** = поток с его личными:
   1.1) значениями **переменных** (промежуточные вычисления, **регистры**)
   1.2) номером **workItem** (номер задачи)
   1.3) с его принадлежностью **workGroup** (а на уровне железа - **warp** и **SM**)
2) **Машинка с 32 лилипутами** = **warp:**
   2.1) есть общий **instruction pointer**
   2.2) может быть **active** - все данные подгружены в регистры - либо сейчас считаем у **станка**-**ALU**, либо ждем когда он освободится
   2.3) может быть **inactive** - ждет когда **транзакции** подгрузят все запрошенные **cache lines** из **VRAM**
3) **Официант** = шина коммуникации с **VRAM**:
   3.1) принимает запросы от **warp**-ов и обслуживает их
   3.2) обладает большой **latency**
   3.3) может одновременно обслуживать много запросов и за счет этого большая общая пропускная способность (общее число приносимых **cache lines**)
4) **Станки**-**ALU = FP32 + INT32** счеты
5) **Клоун** какую играет роль**???**

Мысленная модель, стандарт и API, железо

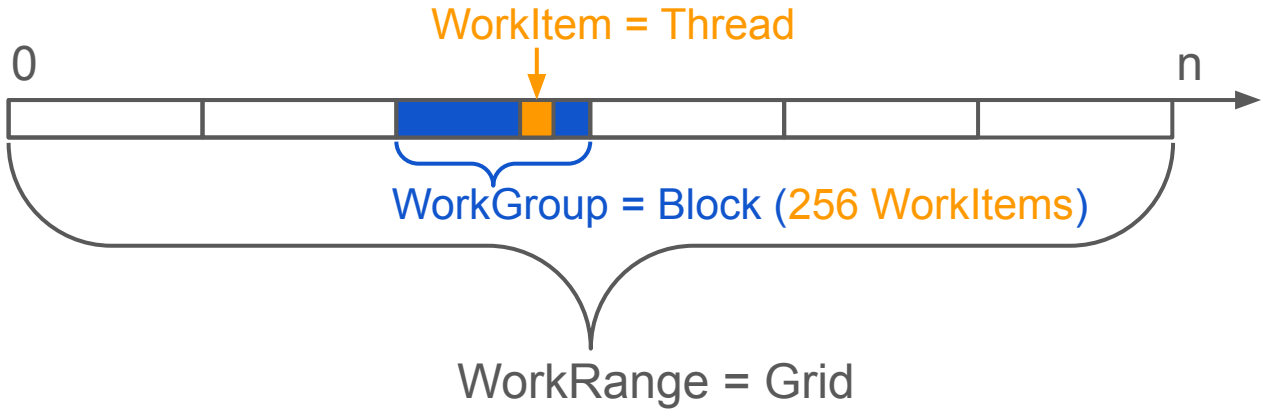
1) **Лилипут** = поток с его личными:
   1.1) значениями **переменных** (промежуточные вычисления, **регистры**)
   1.2) номером **workItem** (номер задачи)
   1.3) с его принадлежностью **workGroup** (а на уровне железа - **warp** и **SM**)
2) **Машинка с 32 лилипутами** = **warp:**
   2.1) есть общий **instruction pointer**
   2.2) может быть **active** - все данные подгружены в регистры - либо сейчас считаем у **станка**-**ALU**, либо ждем когда он освободится
   2.3) может быть **inactive** - ждет когда **транзакции** подгрузят все запрошенные **cache lines** из **VRAM**
3) **Официант** = шина коммуникации с **VRAM**:
   3.1) принимает запросы от **warp**-ов и обслуживает их
   3.2) обладает большой **latency**
   3.3) может одновременно обслуживать много запросов и за счет этого большая общая пропускная способность (общее число приносимых **cache lines**)
4) **Станки**-**ALU = FP32 + INT32** счеты
5) **Клоун** жонглирует **машинками с 32 лилипутами** - вовремя подносит **active** машинку к **станку**-**ALU** (чтобы повысить утилизацию, скрыть **latency**)

Мысленная модель, стандарт и API, железо

1) **Лилипут** = поток с его личными: **минималистичен, прост, ограничен**
   1.1) значениями **переменных** (промежуточные вычисления, **регистры**)
   1.2) номером **workItem** (номер задачи)
   1.3) с его принадлежностью **workGroup** (а на уровне железа - **warp** и **SM**)
2) **Машинка с 32 лилипутами** = **warp:**
   2.1) есть общий **instruction pointer**
   2.2) может быть **active** - все данные подгружены в регистры - либо сейчас считаем у **станка**-**ALU**, либо ждем когда он освободится
   2.3) может быть **inactive** - ждет когда **транзакции** подгрузят все запрошенные **cache lines** из **VRAM**
3) **Официант** = шина коммуникации с **VRAM**:
   3.1) принимает запросы от **warp**-ов и обслуживает их
   3.2) обладает большой **latency**
   3.3) может одновременно обслуживать много запросов и за счет этого большая общая пропускная способность (общее число приносимых **cache lines**)
4) **Станки**-**ALU = FP32 + INT32** счеты
5) **Клоун** жонглирует **машинками с 32 лилипутами** - вовремя подносит **active** машинку к **станку**-**ALU** (чтобы повысить утилизацию, скрыть **latency**)

Мысленная модель, стандарт и API, железо

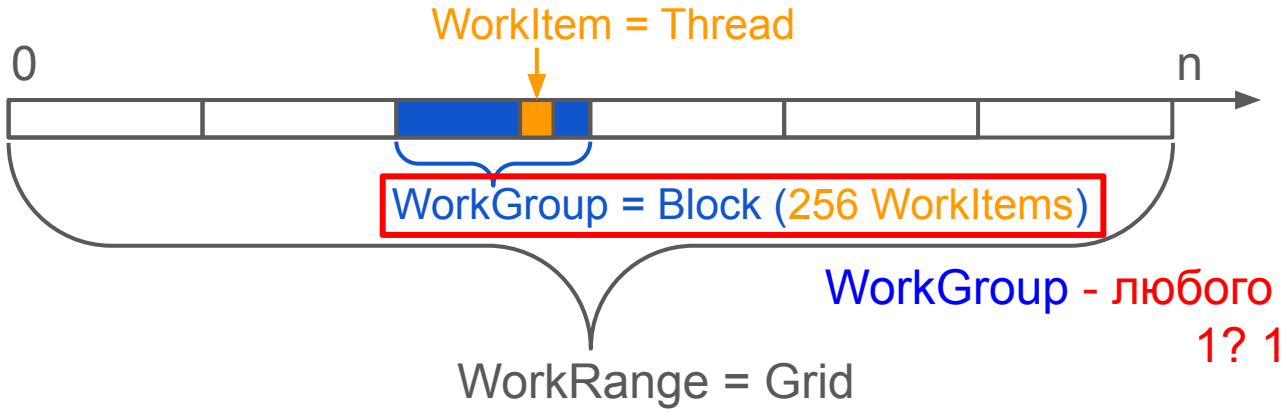
минималистичен, прост, ограничен



+ у каждой WorkGroup есть Local Memory

Мысленная модель, стандарт и API, железо

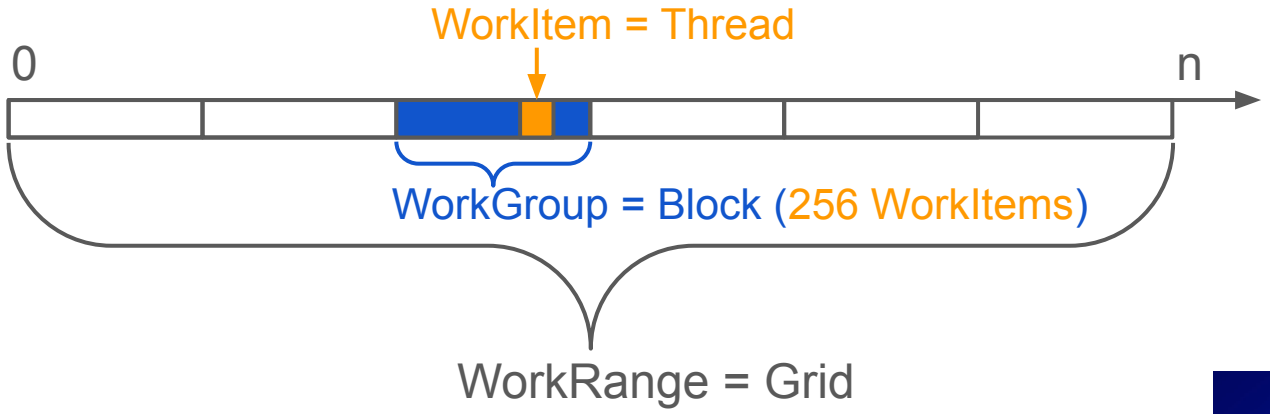
минималистичен, прост, ограничен



+ у каждой WorkGroup есть Local Memory

Мысленная модель, стандарт и API, железо

минималистичен, прост, ограничен



Но из-за простоты не потеряли ли мы мощь **SIMD**?

SIMT (thread) vs **SIMD (data)**

```
__m256i maskAll = _mm256_setzero_si256();
__m256i iters = _mm256_setzero_si256();
__m256 xs = _mm256_set1_ps(0.0f);
__m256 ys = _mm256_set1_ps(0.0f);
for (iteration = 0; iteration < MAX_ITERATIONS; iteration++) {
    __m256 xsn = _mm256_add_ps(_mm256_mul_ps(_mm256_mul_ps(xs, xs), _mm256_mul_ps(ys, ys)), xs0);
    __m256 ysn = _mm256_add_ps(_mm256_mul_ps(_mm256_mul_ps(_mm256_set1_ps(2.0f), xs), ys), _mm256_set1_ps
    xs = _mm256_add_ps(_mm256_andnot_ps((__m256) maskAll, xsn), _mm256_and_ps((__m256) maskAll, xs));
    ys = _mm256_add_ps(_mm256_andnot_ps((__m256) maskAll, ysn), _mm256_and_ps((__m256) maskAll, ys));

    maskAll = (__m256i) _mm256_or_ps(_mm256_cmp_ps(_mm256_add_ps(_mm256_mul_ps(xs, xs), _mm256_mul_ps(ys,
    iters = _mm256_add_epi16(iters, _mm256_andnot_si256(maskAll, _mm256_set1_epi16(1)));
    int mas
    if (mas == (int) 0xffffffff) {
        break;
    }
}
```

short b0 | short b1 | short b2 | short b3

+

short c0 | short c1 | short c2 | short c3

What is ISPC?

`ispc` is a compiler for a variant of the C programming language, with extensions for "single program, multiple data" (SPMD) programming. Under the SPMD model, the programmer writes a program that generally appears to be a regular serial program, though the execution model is actually that a number of *program instances* execute in parallel on the hardware. (See the ispc documentation for more details and examples that illustrate this concept.)

Performance

`ispc` compiles a C-based SPMD programming language to run on the SIMD units of CPUs and GPUs; it frequently provides a 3x or more speedup on architectures with 4-wide vector SSE units and 5x-6x on architectures with 8-wide AVX vector units, without any of the difficulty of writing intrinsics code. Parallelization across multiple cores is also supported by `ispc`, making it possible to write programs that achieve performance improvement that scales by both number of cores and vector unit size.

Resources

GitHub Repository

Community Discussions

Report Issues

Release Planning

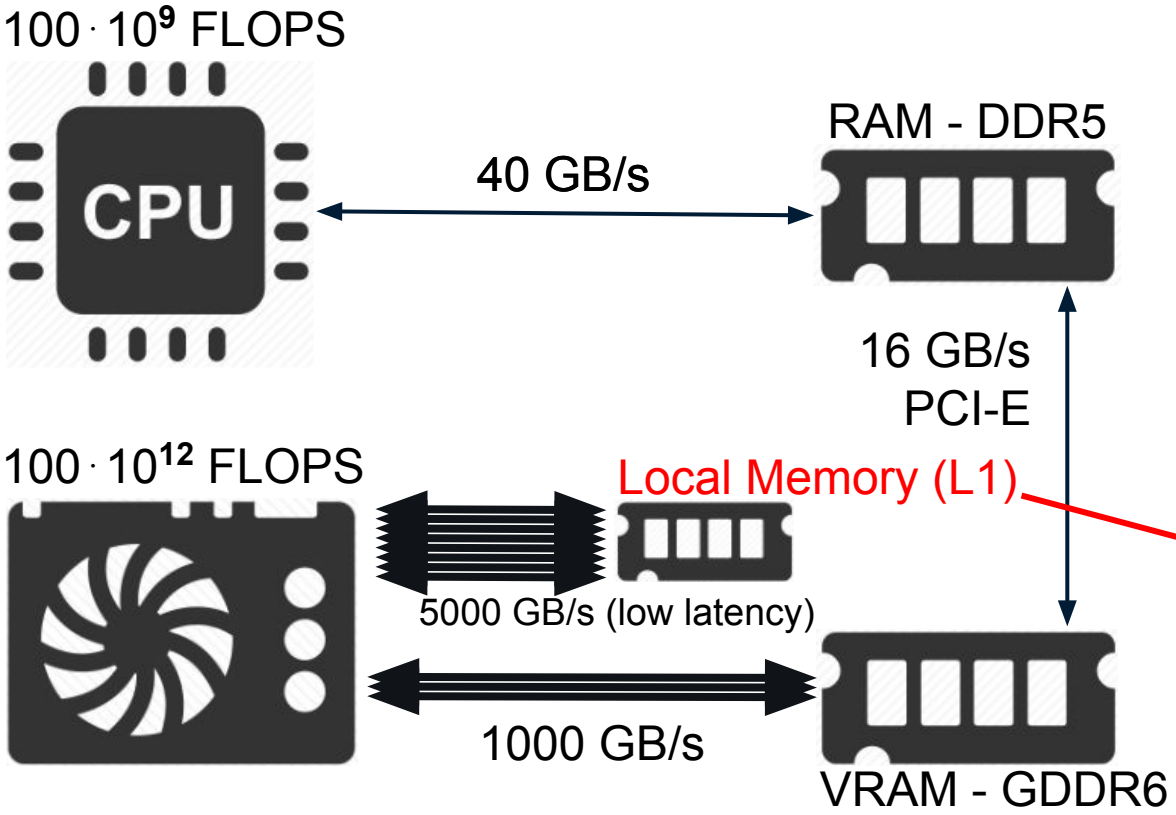
Contributing Guide

Documentation Wiki

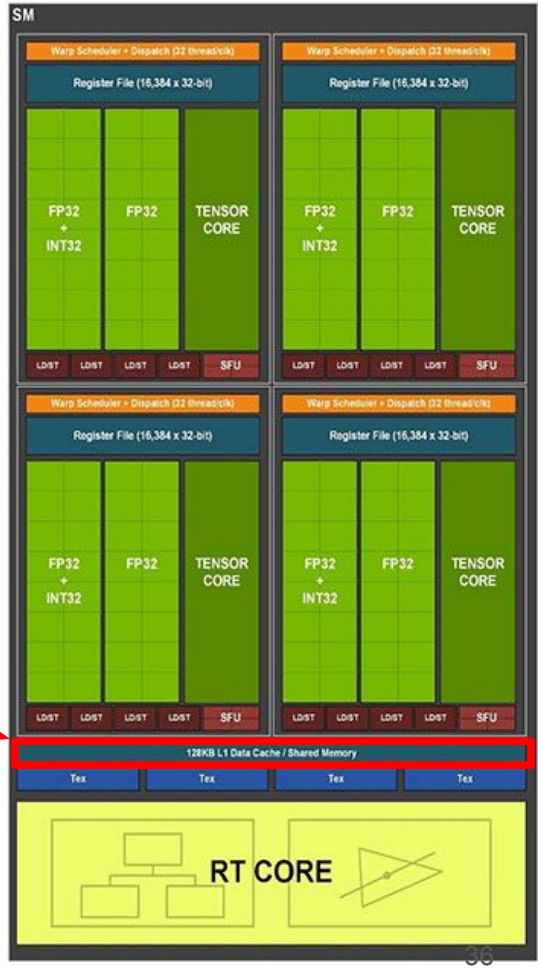
Глава 3: локальная память

Local/Shared memory, barriers, bank conflicts

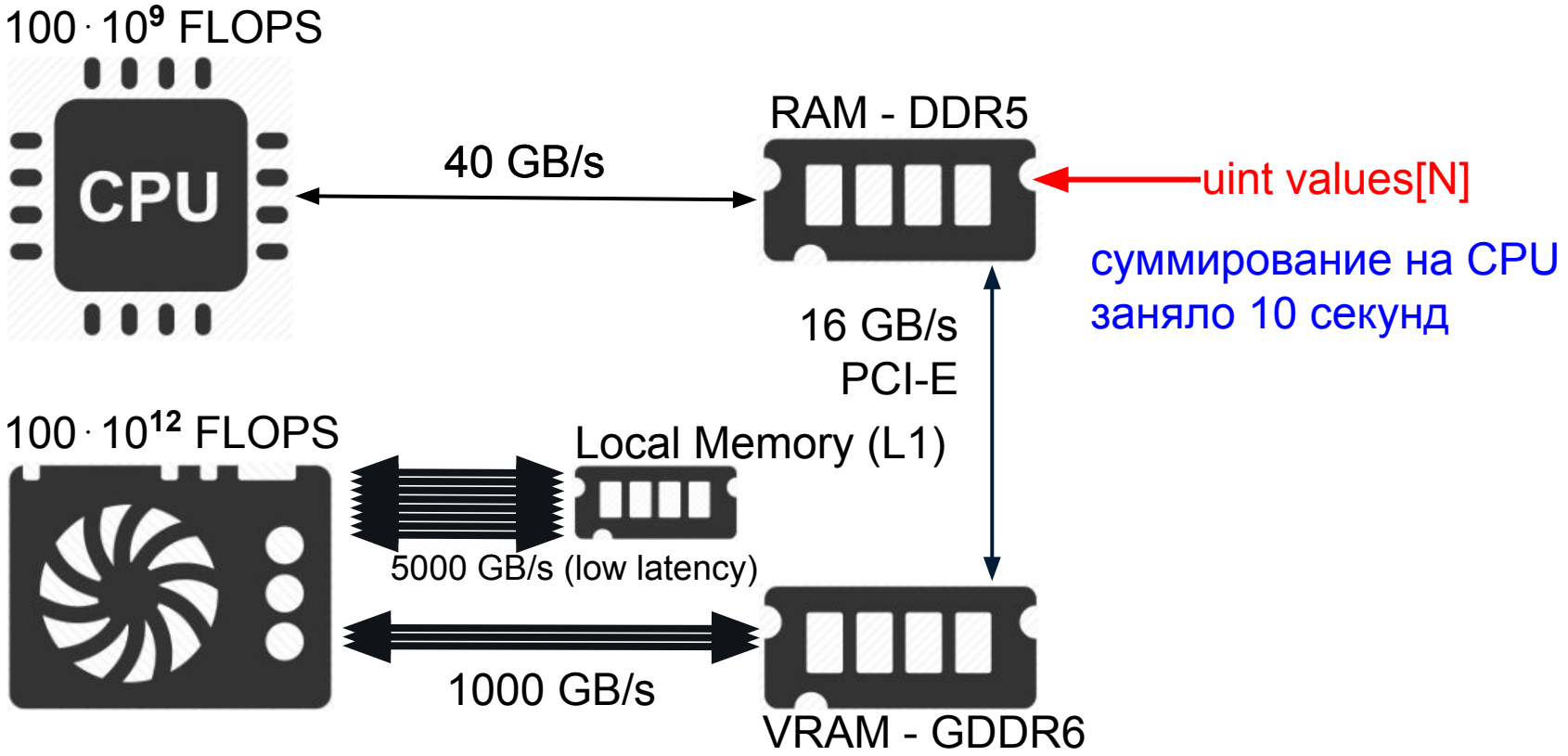
Локальная память



SM - Streaming Multiprocessor

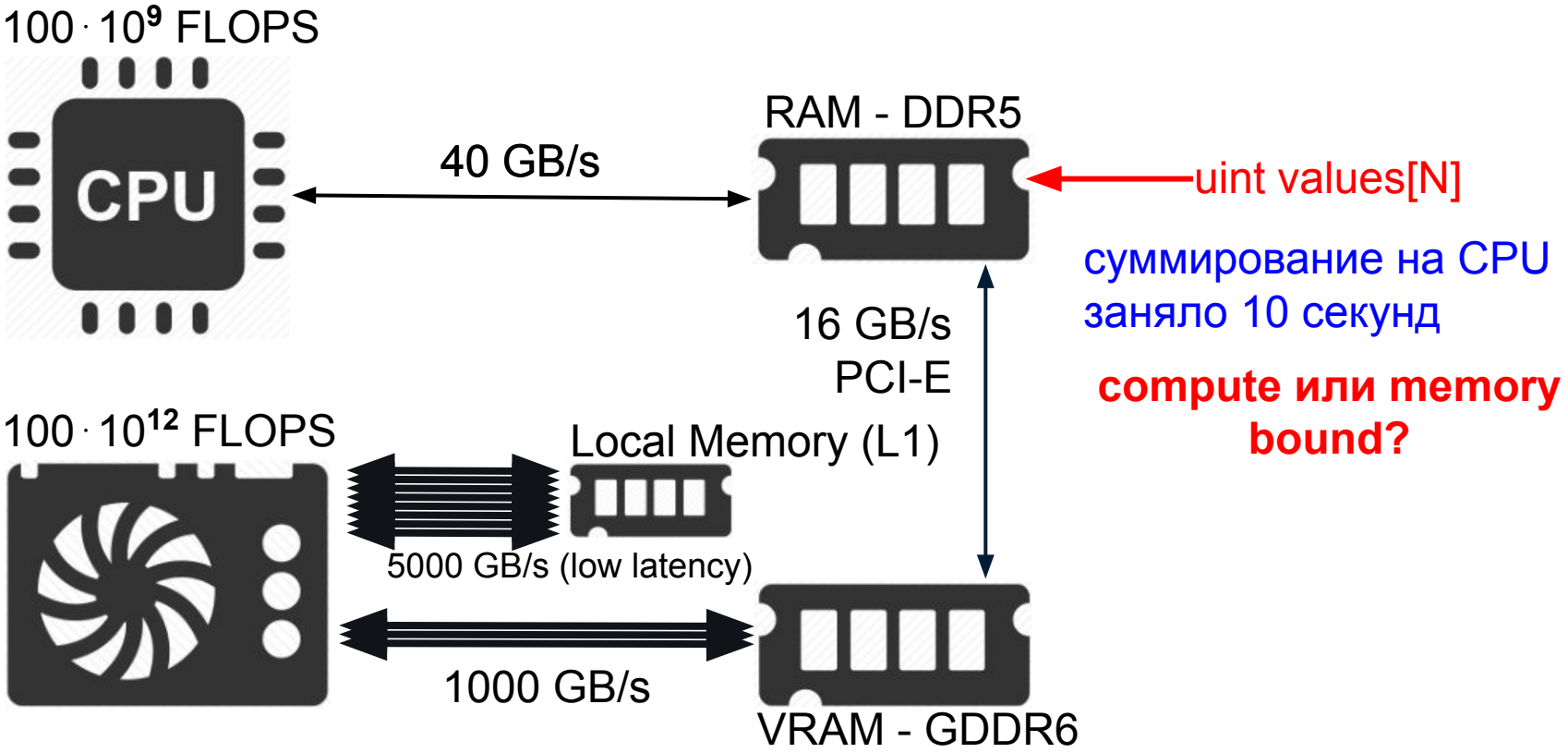


Локальная память

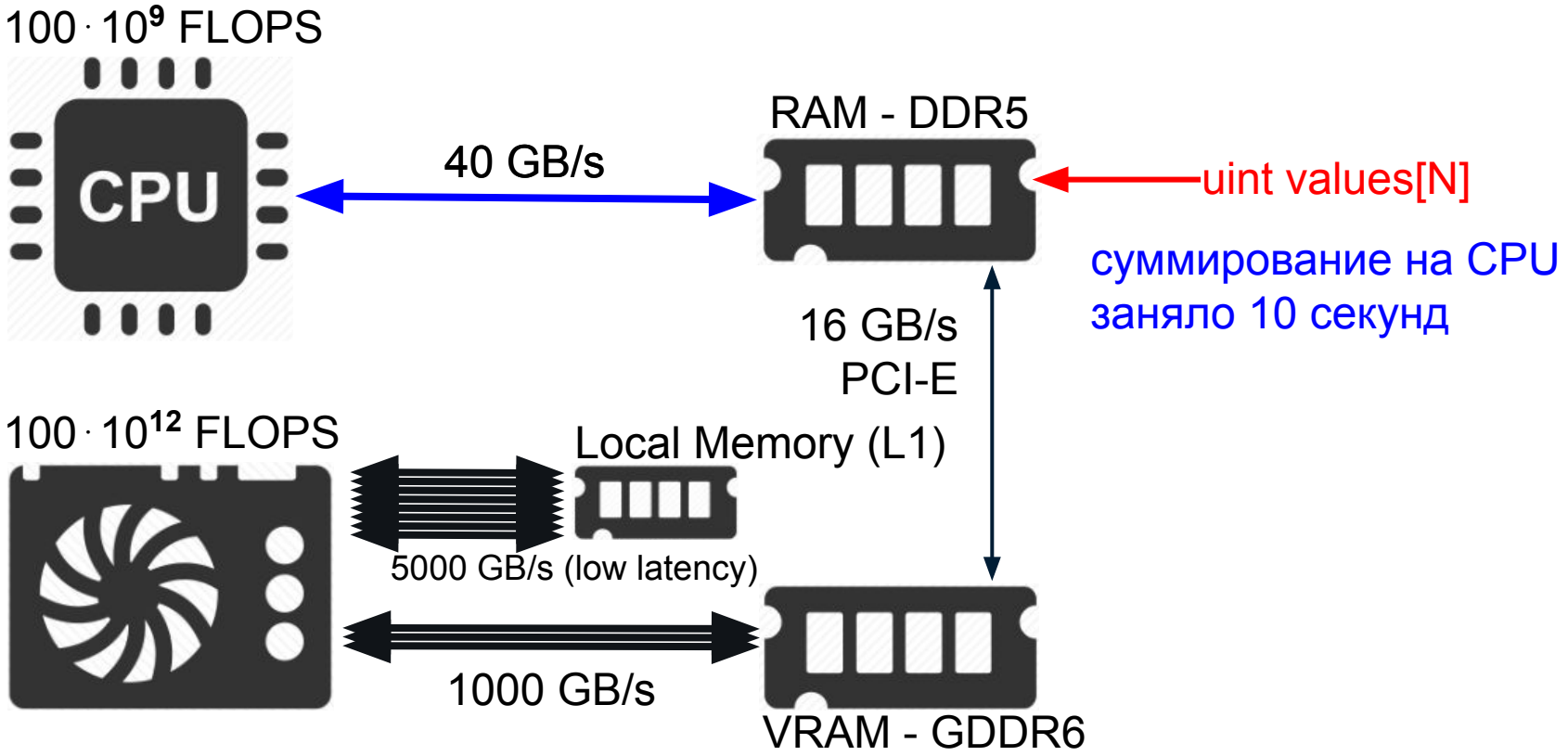



Все зависит от узкого места!

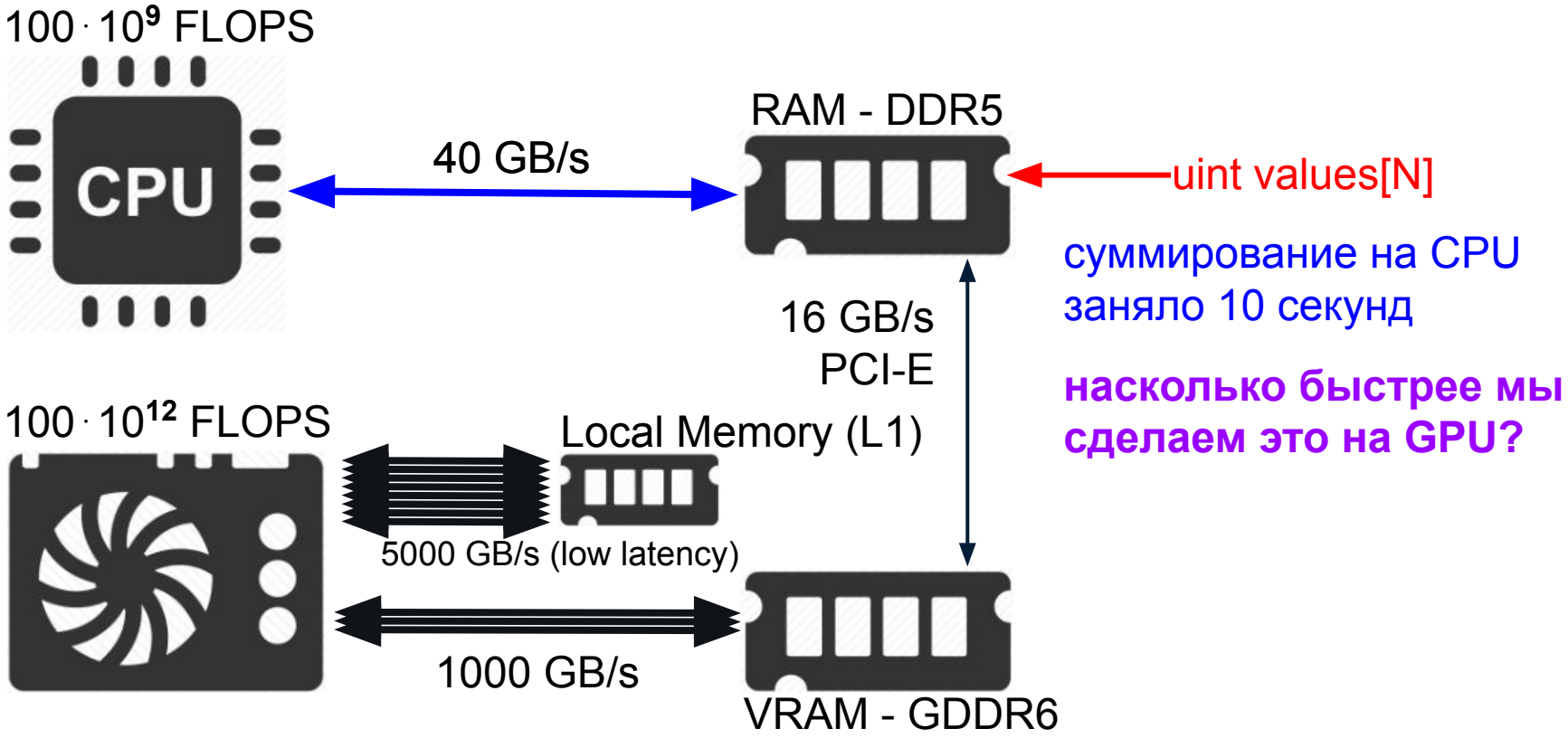
Локальная память



Локальная память

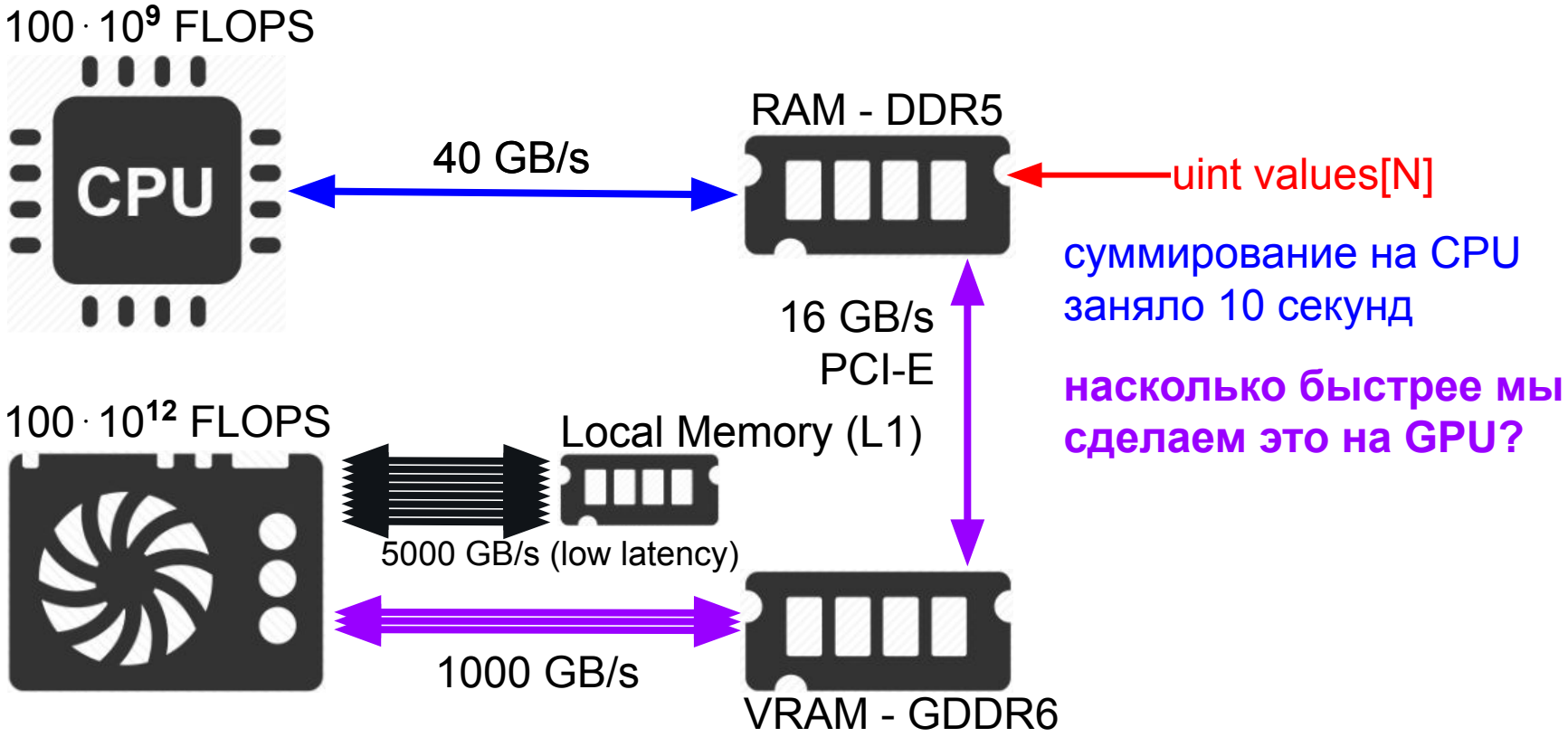


Локальная память

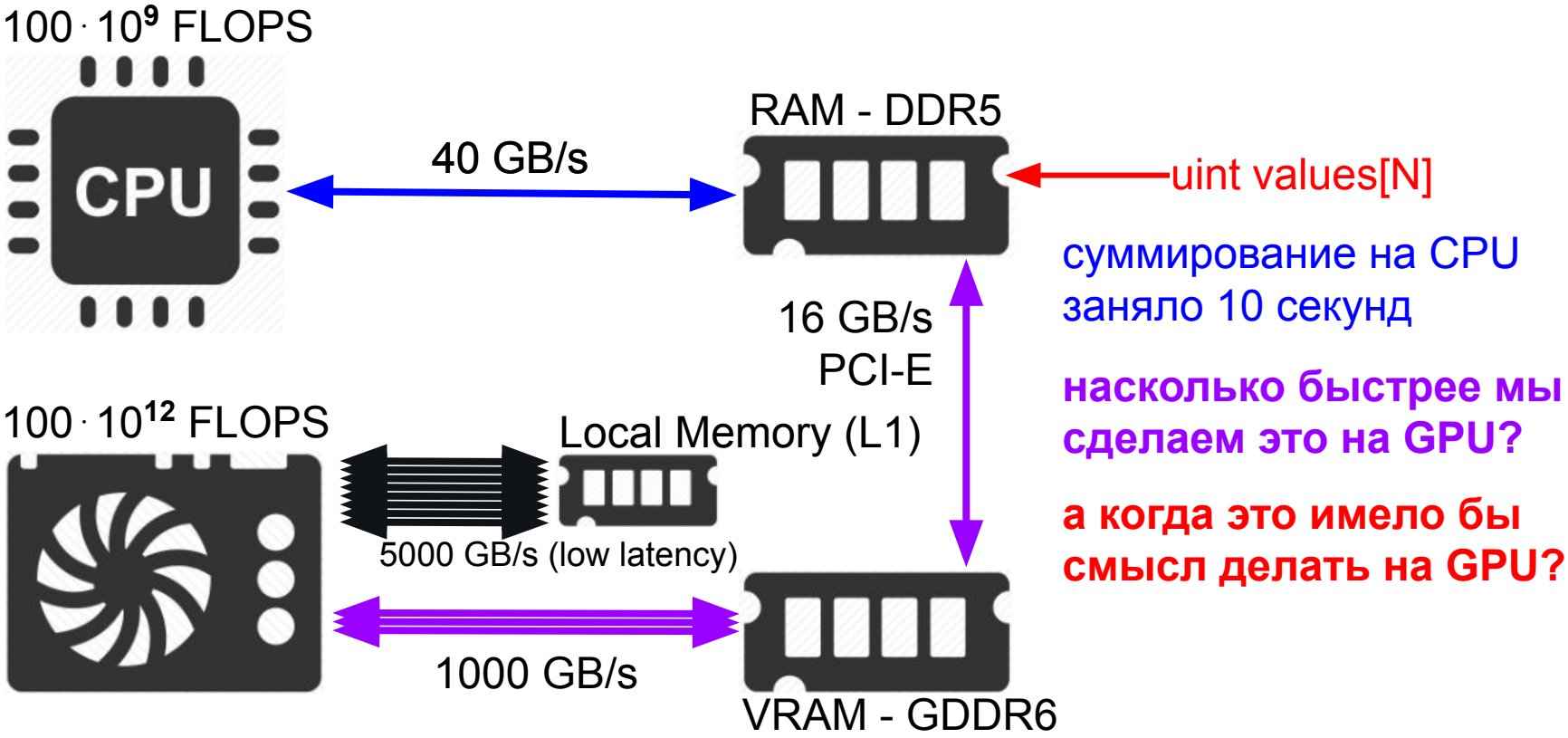



Все зависит от узкого места!

Локальная память



Локальная память



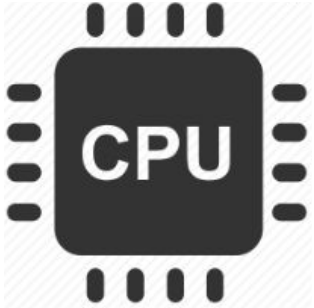
суммирование на CPU заняло 10 секунд

насколько быстрее мы сделаем это на GPU?

а когда это имело бы смысл делать на GPU?

Локальная память

$100 \cdot 10^9$ FLOPS



40 GB/s

RAM - DDR5

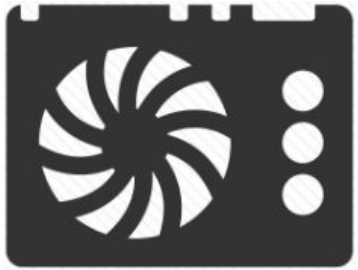
uint values[N]

суммирование на CPU заняло 10 секунд

16 GB/s
PCI-E

насколько быстрее мы сделаем это на GPU?

$100 \cdot 10^{12}$ FLOPS



Local Memory (L1)

5000 GB/s (low latency)

а когда это имело бы смысл делать на GPU?

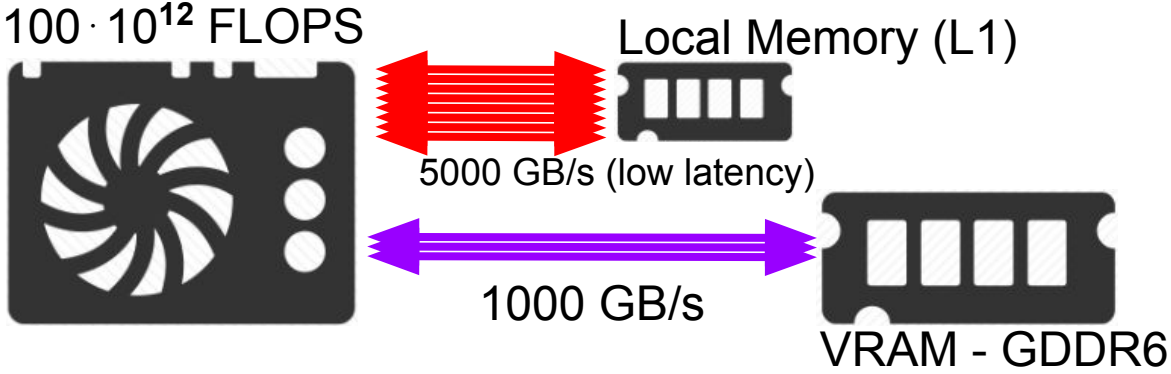
1000 GB/s

VRAM - GDDR6

а когда имеет смысл использовать Local Memory?

Локальная память

Всегда ли можно грузить **coalesced**?

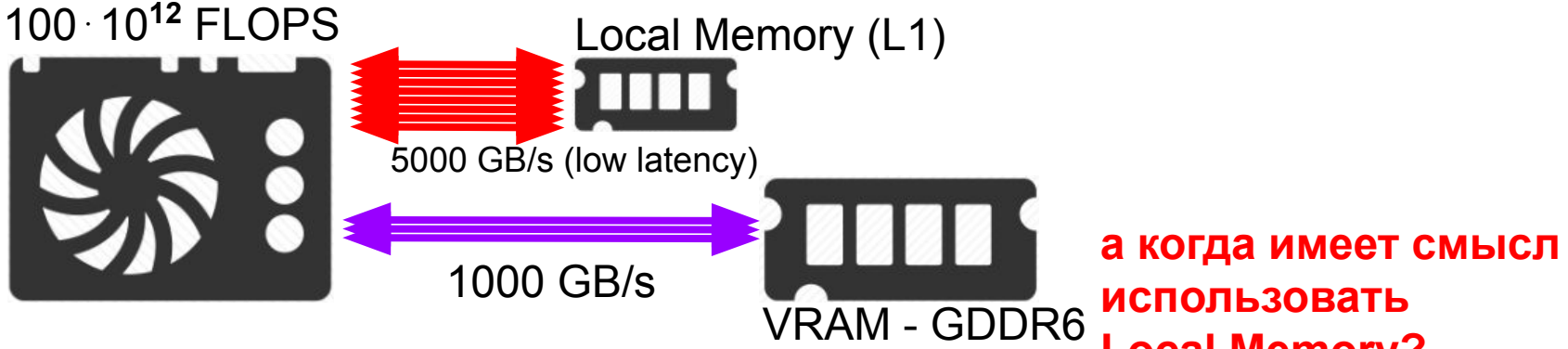


а когда имеет смысл использовать Local Memory?

Локальная память

Всегда ли можно грузить **coalesced**?

Подумайте про транспонирование матрицы!
Чтение - coalesced? Запись - coalesced?

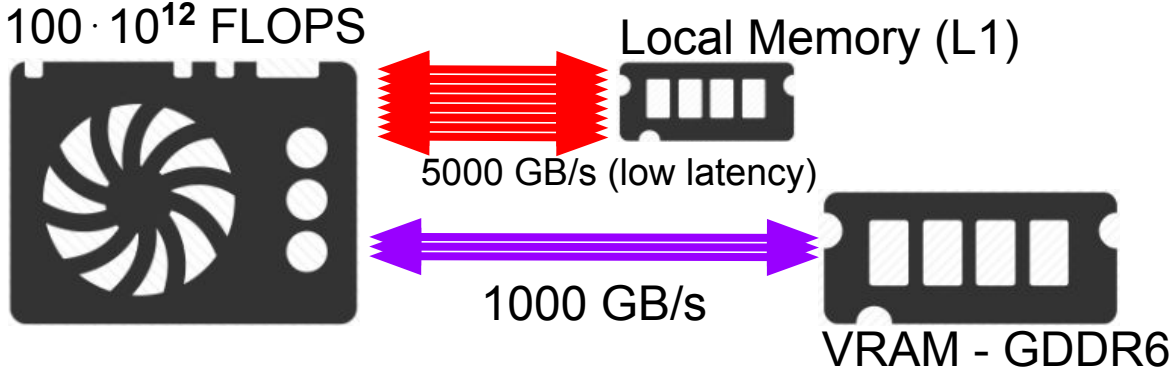


а когда имеет смысл использовать Local Memory?

Локальная память

Всегда ли можно грузить **coalesced**?

А если у нас **A**rray-**o**f-**S**tructures? **(AoS)**



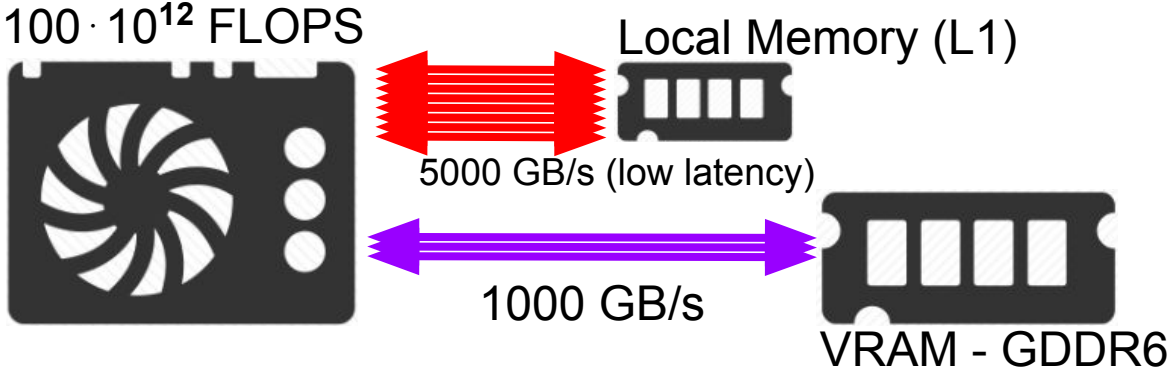
а когда имеет смысл использовать Local Memory?

Локальная память

Всегда ли можно грузить **coalesced**?

А если у нас **A**rray-**o**f-**S**tructures? **(AoS)**

Всегда ли элемент грузится **один раз**?



а когда имеет смысл использовать Local Memory?

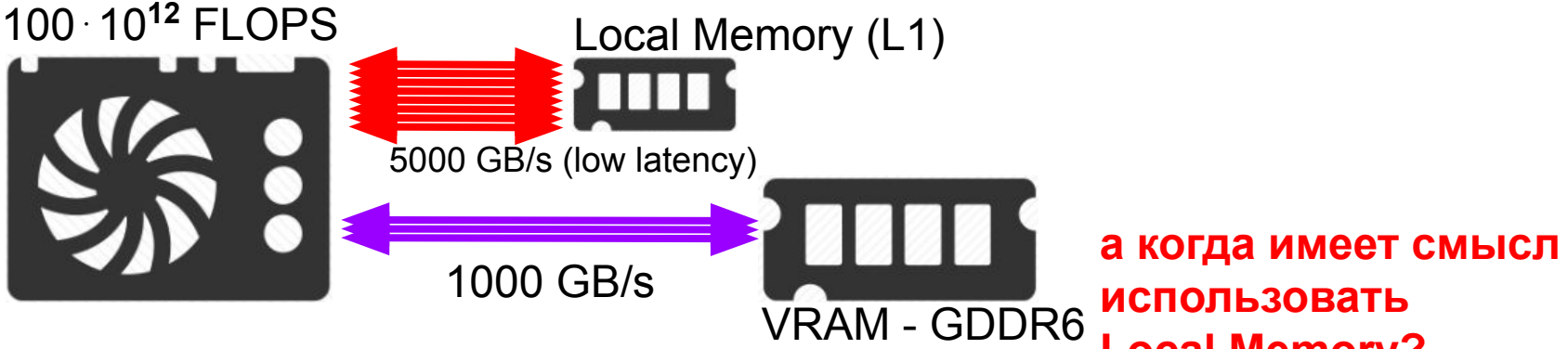
Локальная память

Всегда ли можно грузить **coalesced**?

А если у нас **A**rray-**o**f-**S**tructures? **(AoS)**

Всегда ли элемент грузится **один раз**?

N-body simulation!



а когда имеет смысл использовать Local Memory?

Локальная память

1) Как объявить?

```
__local unsigned int workgroup_data[256];
__shared__ unsigned int workgroup_data[256];
shared uint workgroup_data[256];
```

Локальная память

1) Как объявить?

```
__local unsigned int workgroup_data[256];
__shared__ unsigned int workgroup_data[256];
shared uint workgroup_data[256];
```

2) Как выглядит:

```
__local unsigned int workgroup_data[256];
if (index < n) {
    workgroup_data[get_local_id(0)] = a[index];
} else {
    workgroup_data[get_local_id(0)] = 0;
}
```

Аллокация

Локальная память

1) Как объявить?

```
__local unsigned int workgroup_data[256];
__shared__ unsigned int workgroup_data[256];
shared uint workgroup_data[256];
```

2) Как выглядит:

```
__local unsigned int workgroup_data[256];
if (index < n) {
    workgroup_data[get_local_id(0)] = a[index];
} else {
    workgroup_data[get_local_id(0)] = 0;
}
```

Аллокация

Чтение своего элемента

Локальная память

1) Как объявить?

```
__local unsigned int workgroup_data[256];
__shared__ unsigned int workgroup_data[256];
shared uint workgroup_data[256];
```

2) Как выглядит:

```
__local unsigned int workgroup_data[256];
if (index < n) {
    workgroup_data[get_local_id(0)] = a[index];
} else {
    workgroup_data[get_local_id(0)] = 0;
}
```

Аллокация

Чтение своего элемента

Чтение нейтрального элемента (**padding**)

Локальная память

1) Как объявить?

```
__local unsigned int workgroup_data[256];
__shared__ unsigned int workgroup_data[256];
shared uint workgroup_data[256];
```

2) Как выглядит:

```
__local unsigned int workgroup_data[256];
if (index < n) {
    workgroup_data[get_local_id(0)] = a[index];
} else {
    workgroup_data[get_local_id(0)] = 0;
}
```

Аллокация

Чтение своего элемента

Чтение нейтрального элемента (**padding**)

Может ли теперь поток **warp** прочитать число соседа?

Локальная память

1) Как объявить?

```
__local unsigned int workgroup_data[256];
__shared__ unsigned int workgroup_data[256];
shared uint workgroup_data[256];
```

2) Как выглядит:

```
__local unsigned int workgroup_data[256];
if (index < n) {
    workgroup_data[get_local_id(0)] = a[index];
} else {
    workgroup_data[get_local_id(0)] = 0;
}
```

Аллокация

Чтение своего элемента

Чтение нейтрального элемента (**padding**)

Может ли теперь поток **workGroup** прочитать число соседа?

Локальная память

1) Как объявить?

```
__local unsigned int workgroup_data[256];
__shared__ unsigned int workgroup_data[256];
shared uint workgroup_data[256];
```

2) Как выглядит:

```
__local unsigned int workgroup_data[256];
if (index < n) {
    workgroup_data[get_local_id(0)] = a[index];
} else {
    workgroup_data[get_local_id(0)] = 0;
}
barrier(CLK_LOCAL_MEM_FENCE);
```

Аллокация

Чтение своего элемента

Чтение нейтрального элемента (**padding**)

Синхронизация **workGroup**

Локальная память

1) Как объявить?

```
__local unsigned int workgroup_data[256];
__shared__ unsigned int workgroup_data[256];
shared uint workgroup_data[256];
```

2) Как выглядит:

```
__local unsigned int workgroup_data[256];
if (index < n) {
    workgroup_data[get_local_id(0)] = a[index];
} else {
    workgroup_data[get_local_id(0)] = 0;
}
barrier(CLK_LOCAL_MEM_FENCE);
```


```
__shared__ unsigned int workgroup_data[256];
if (index < n) {
    workgroup_data[threadIdx.x] = a[index];
} else {
    workgroup_data[threadIdx.x] = 0;
}
__syncthreads();
```

Локальная память - bank conflicts

```
__local unsigned int workgroup_data[256];
workgroup_data[get_local_id(0)] = a[index];
```

coalesced memory access?

Локальная память - bank conflicts

```
__local unsigned int workgroup_data[256];
workgroup_data[get_local_id(0)] = a[index];
```

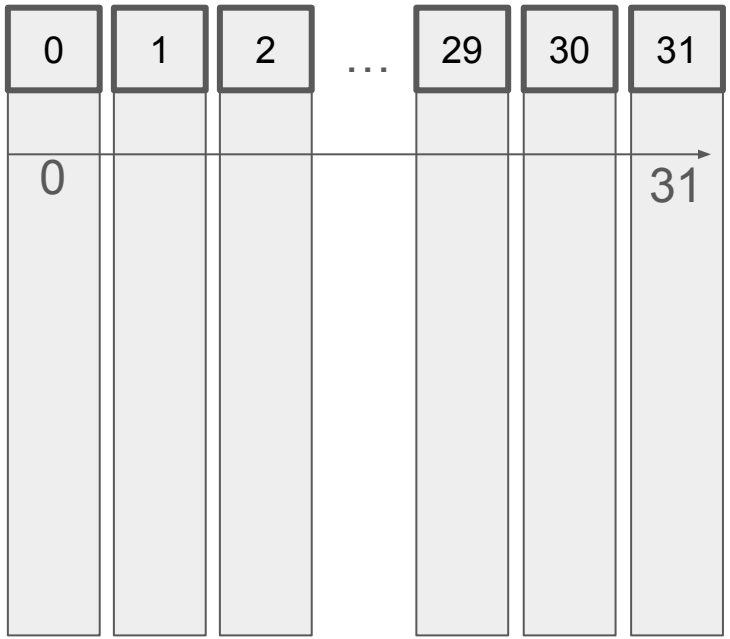
Local memory banks:

Локальная память - bank conflicts

```
__local unsigned int workgroup_data[256];
workgroup_data[get_local_id(0)] = a[index];
```

Local memory banks:

Local address space:

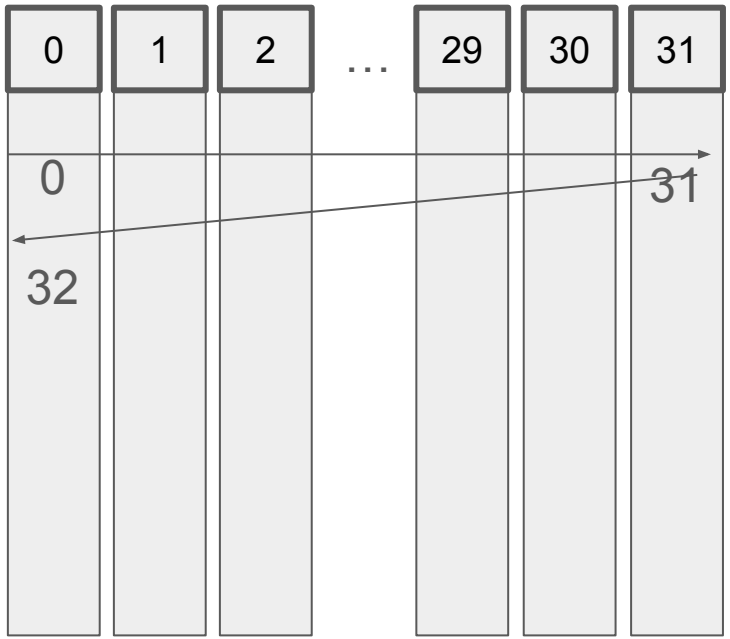


Локальная память - bank conflicts

```
__local unsigned int workgroup_data[256];
workgroup_data[get_local_id(0)] = a[index];
```

Local memory banks:

Local address space:

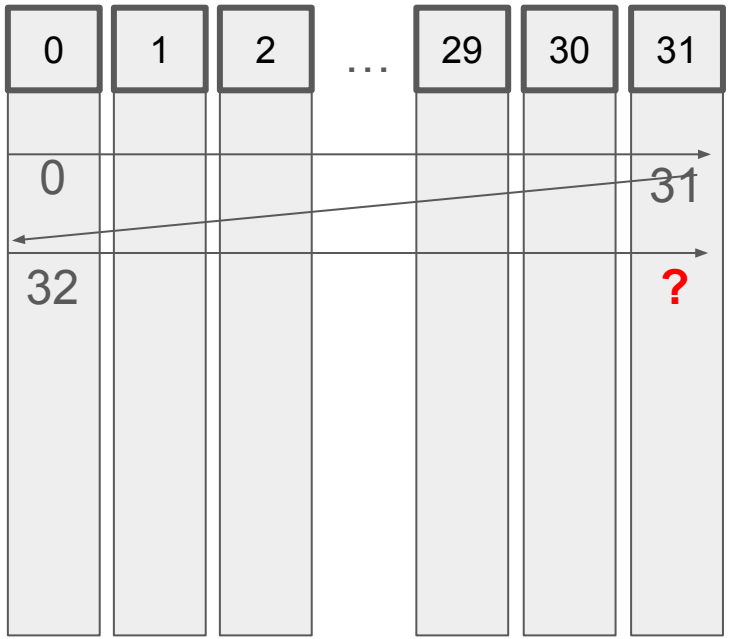


Локальная память - bank conflicts

```
__local unsigned int workgroup_data[256];
workgroup_data[get_local_id(0)] = a[index];
```

Local memory banks:

Local address space:

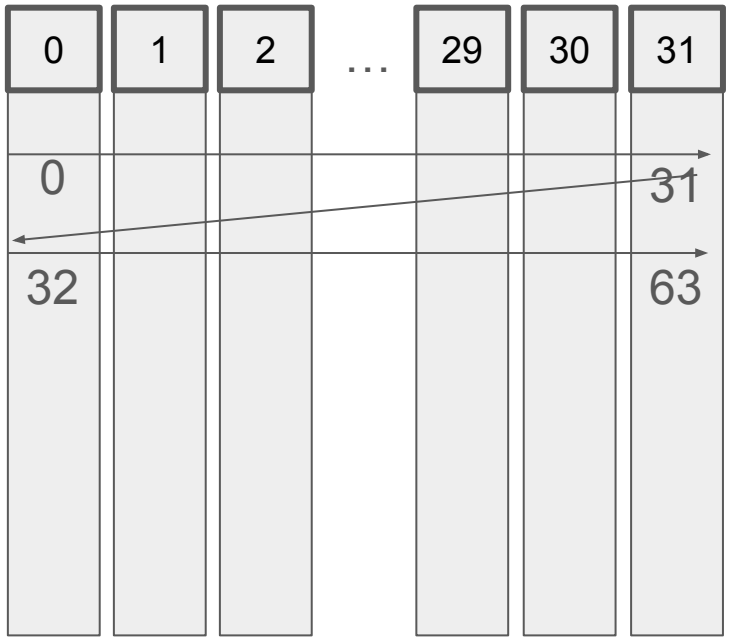


Локальная память - bank conflicts

```
__local unsigned int workgroup_data[256];
workgroup_data[get_local_id(0)] = a[index];
```

Local memory banks:

Local address space:

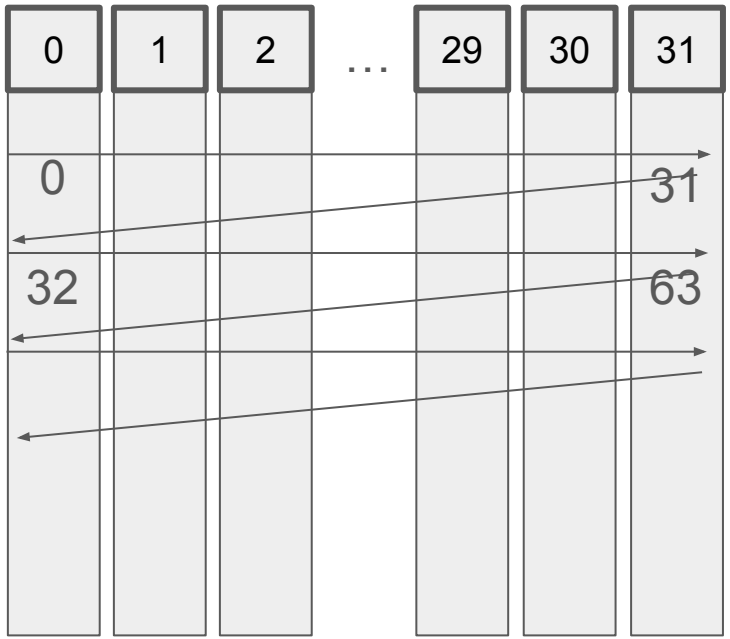


Локальная память - bank conflicts

```
__local unsigned int workgroup_data[256];
workgroup_data[get_local_id(0)] = a[index];
```

Local memory banks:

Local address space:



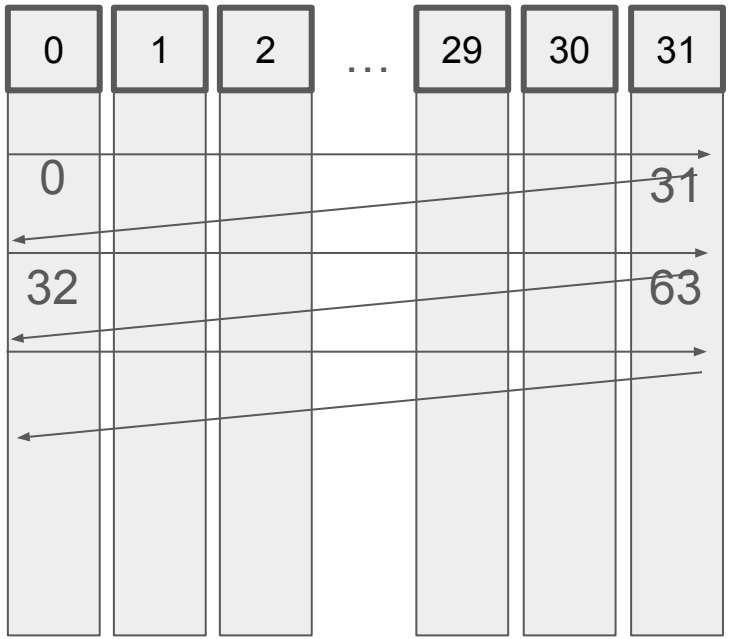
Локальная память - bank conflicts

```
__local unsigned int workgroup_data[256];
workgroup_data[get_local_id(0)] = a[index];
```

Какие banks обрабатывают?

Local memory banks:

Local address space:



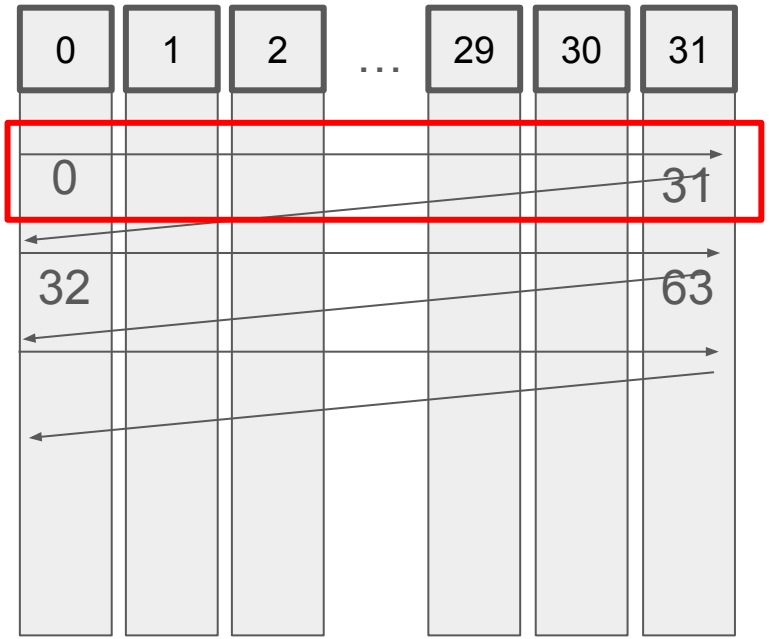
Локальная память - bank conflicts

```
__local unsigned int workgroup_data[256];
workgroup_data[get_local_id(0)] = a[index];
```

Какие banks обрабатывают?

Local memory banks:

Local address space:



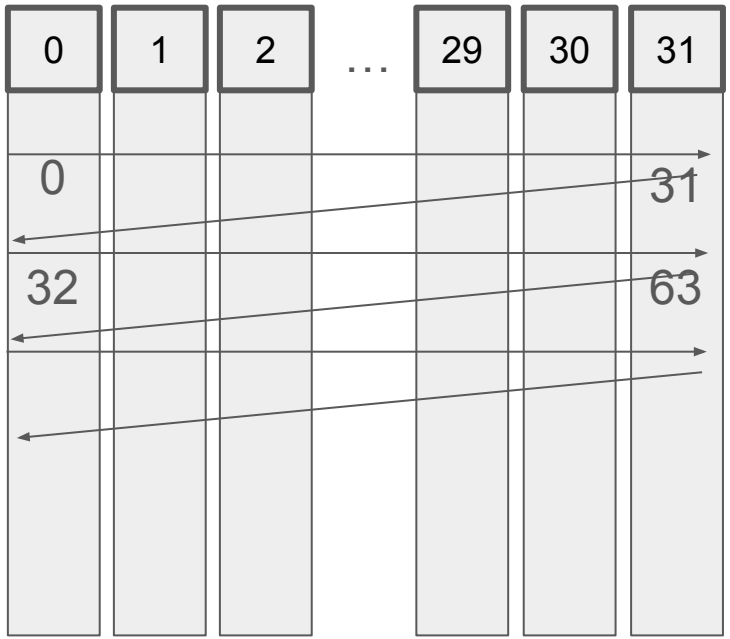
Локальная память - bank conflicts

```
__local unsigned int workgroup_data[256];
workgroup_data[32 * get_local_id(0)] = a[index];
```

Какие banks обрабатывают?

Local memory banks:

Local address space:



Локальная память - bank conflicts

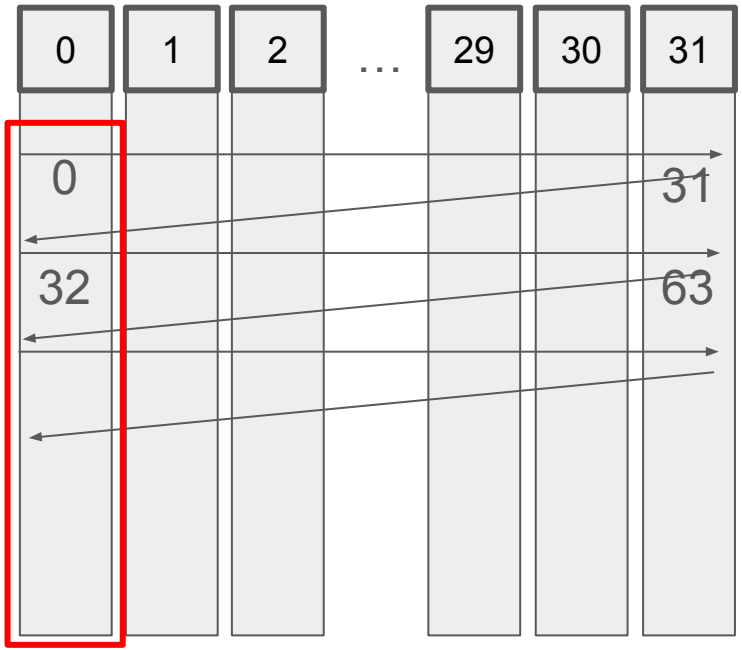
```
__local unsigned int workgroup_data[256];
workgroup_data[32 * get_local_id(0)] = a[index];
```

Какие banks обрабатывают?

Bank conflict!

Local memory banks:

Local address space:



Локальная память - bank conflicts

```
__local unsigned int workgroup_data[256];
workgroup_data[32 * get_local_id(0)] = a[index];
```

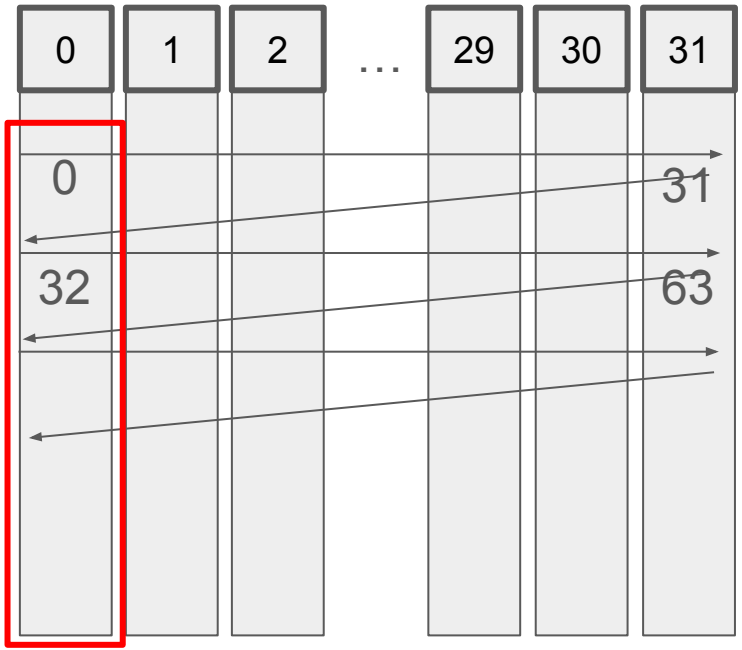
Какие banks обрабатывают?

Bank conflict!

Насколько просели?

Local memory banks:

Local address space:



Локальная память - bank conflicts

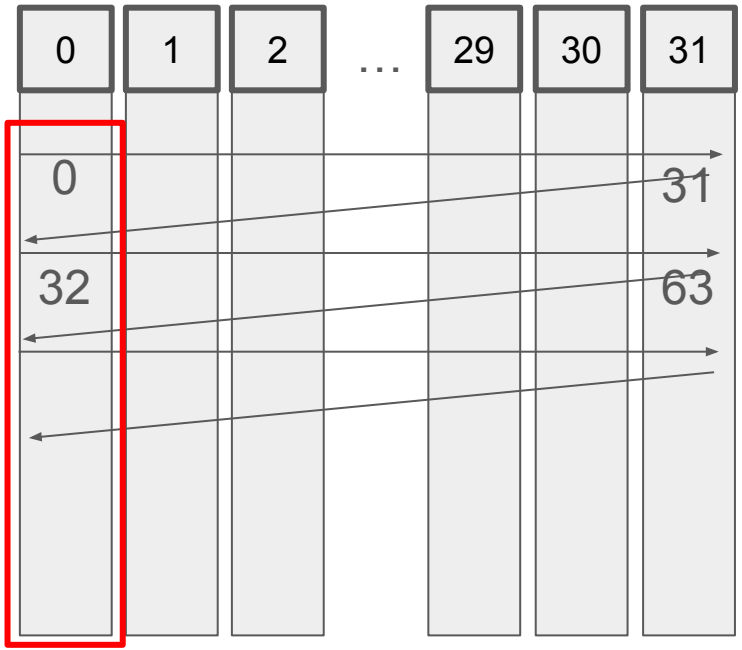
```
__local unsigned int workgroup_data[256];
workgroup_data[32 * get_local_id(0)] = a[index];
```

Какие banks обрабатывают?

Bank conflict!

Local memory banks:

Local address space:



Насколько просели?

А почему мы анализируем **warp,** а не **workGroup?**

Локальная память - bank conflicts

```
__local unsigned int workgroup_data[256];
```

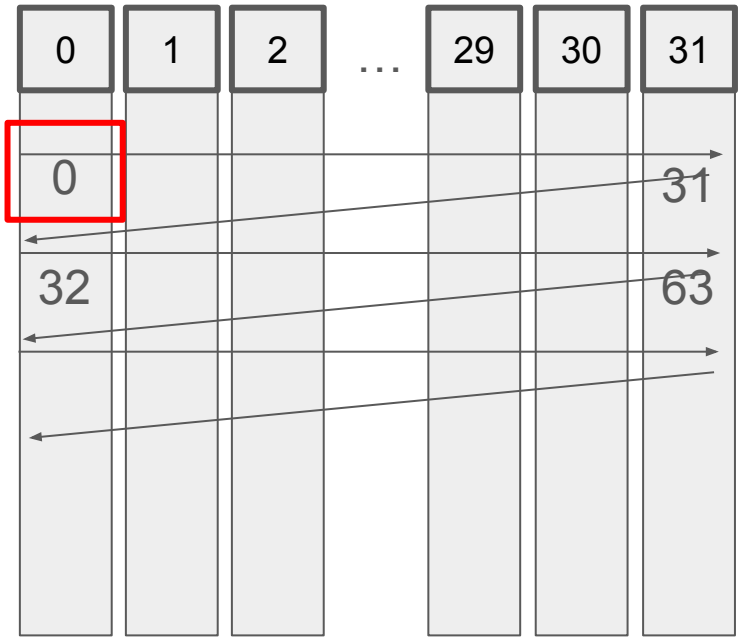
`workgroup_data[0]`

Какие banks обрабатывают?

Быстро или нет?

Local memory banks:

Local address space:



Локальная память - bank conflicts

```
__local unsigned int workgroup_data[256];
```

```
workgroup_data[0]
```

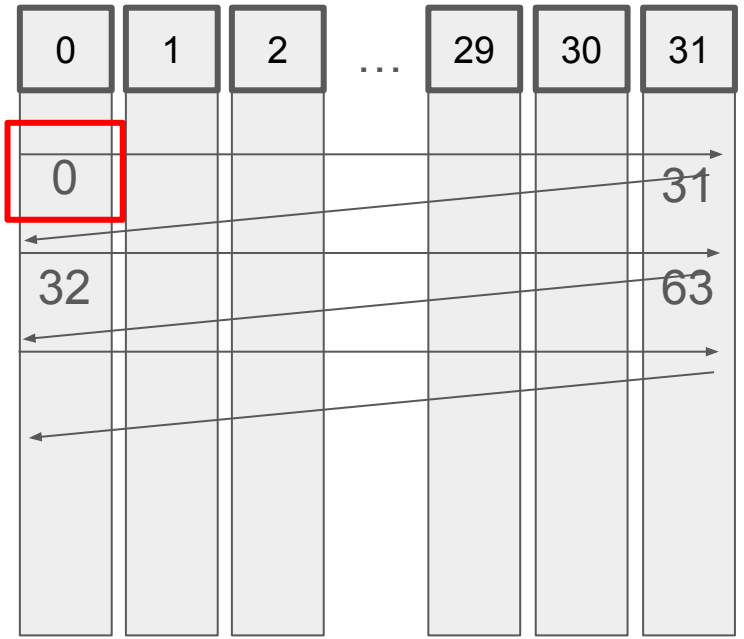
Какие banks обрабатывают?

Быстро или нет?

Broadcast!

Local memory banks:

Local address space:




Перерыв!
Streaming
Pubiprocestor
VIDIA

Глава 4: суммирование

atomicAdd, reduction

01: CPU

```
unsigned int cpu::sum(const unsigned int *values, unsigned int n)
{
    unsigned int sum = 0;
    for (size_t i = 0; i < n; ++i) {
        sum += values[i];
    }
    return sum;
}
```

02: CPU - OpenMP

```
unsigned int cpu::sumOpenMP(const unsigned int *values, unsigned int n)
{
    unsigned int sum = 0;
    #pragma omp parallel for schedule(dynamic, 1024) reduction(+:sum)
    for (ptrdiff_t i = 0; i < n; ++i) {
        sum += values[i];
    }
    return sum;
}
```

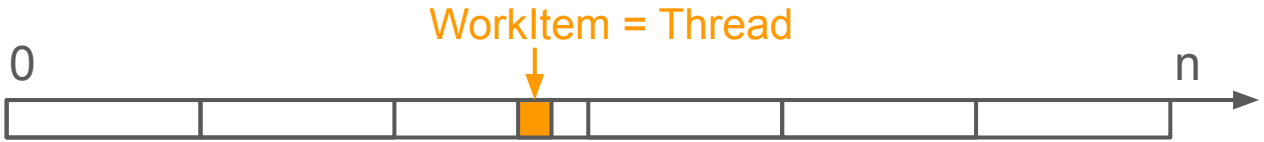
02: CPU - OpenMP

```
unsigned int cpu::sumOpenMP(const unsigned int *values, unsigned int n)
{
    unsigned int sum = 0;
    #pragma omp parallel for schedule(dynamic, 1024) reduction(+:sum)
    for (ptrdiff_t i = 0; i < n; ++i) {
        sum += values[i];
    }
    return sum;
}
```

02: CPU - OpenMP

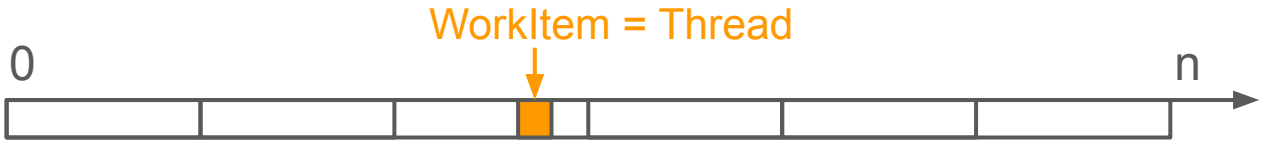
```
unsigned int cpu::sumOpenMP(const unsigned int *values, unsigned int n)
{
    unsigned int sum = 0;
    #pragma omp parallel for schedule(dynamic, 1024) reduction(+:sum)
    for (ptrdiff_t i = 0; i < n; ++i) {
        sum += values[i];
    }
    return sum;
}
```

03: GPU



Что делает каждый **WorkItem**???

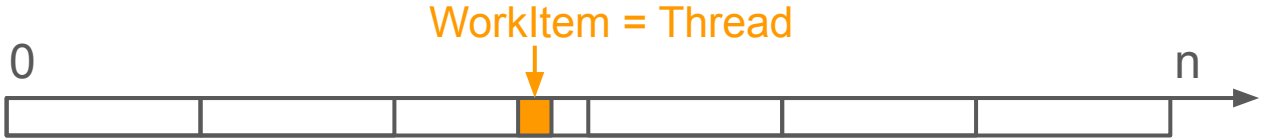
03: GPU



Что делает каждый **WorkItem**???

```
sum[0] += values[index];
```

03: GPU - **atomicAdd**

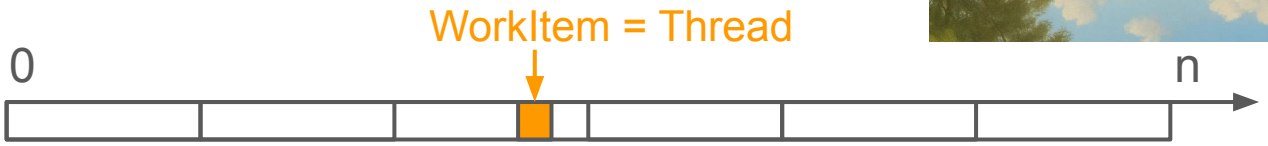


Что делает каждый **WorkItem**???

~~sum[0] += values[index];~~

```
atomicAdd(sum, values[index]);
```

03: GPU - **atomicAdd**



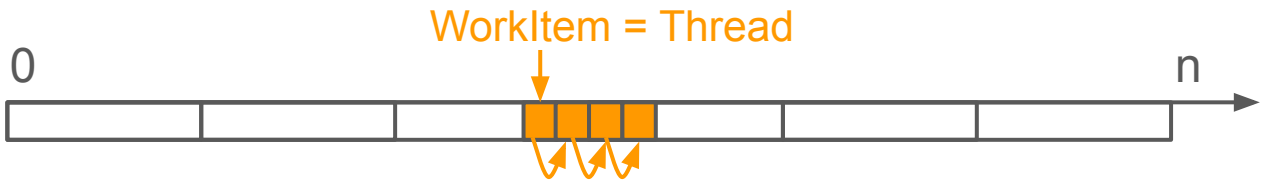
Что делает каждый **WorkItem**???

~~sum[0] += values[index];~~

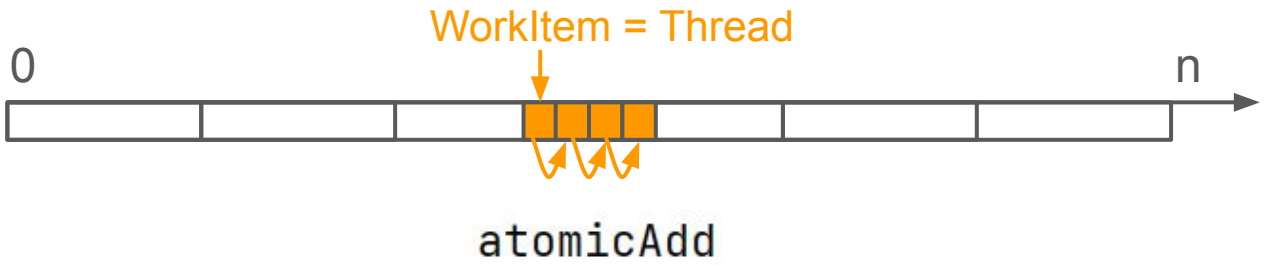
```
atomicAdd(sum, values[index]);
```

Можно ли лучше?

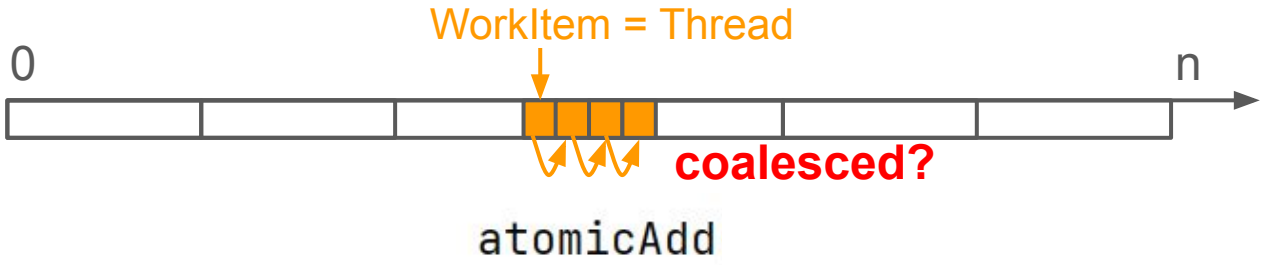
04: GPU - более редкий **atomicAdd**



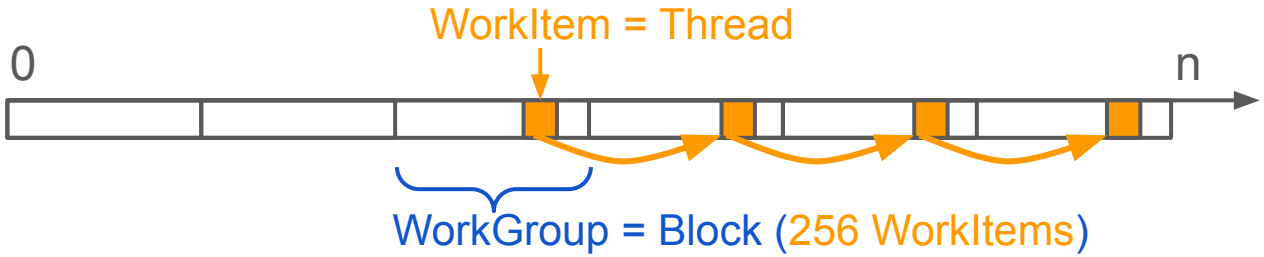
04: GPU - более редкий **atomicAdd**



04: GPU - более редкий **atomicAdd**

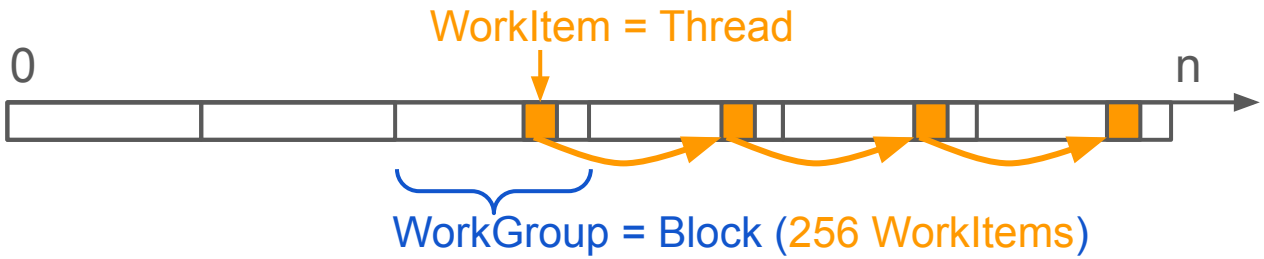


04: GPU - более редкий **atomicAdd**



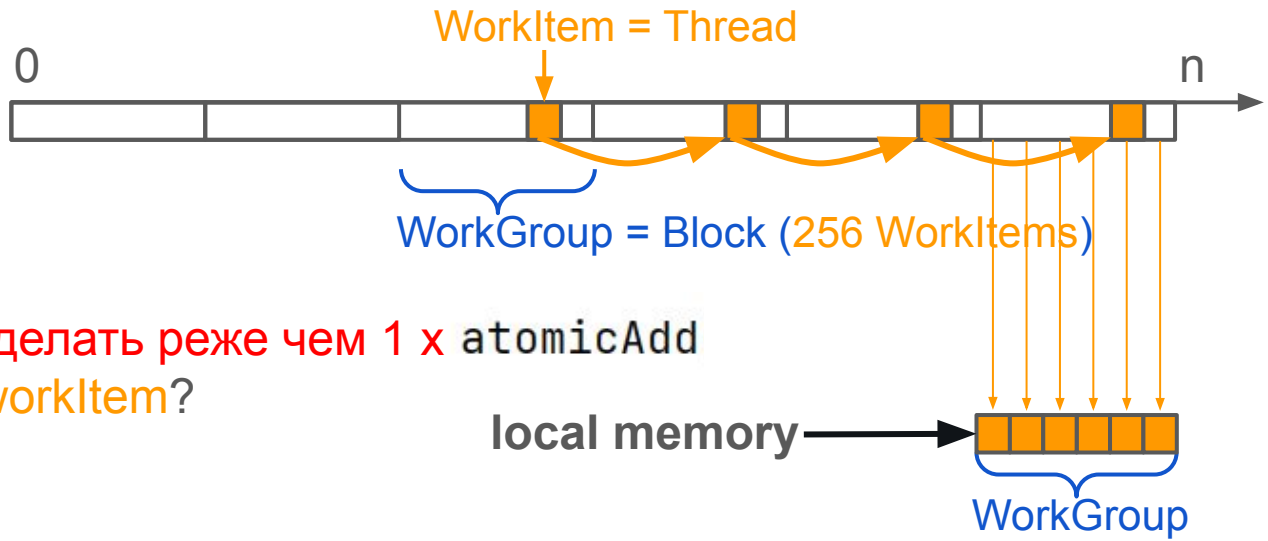

```
atomicAdd
```

05: GPU - **ЕЩЁ** более редкий **atomicAdd**

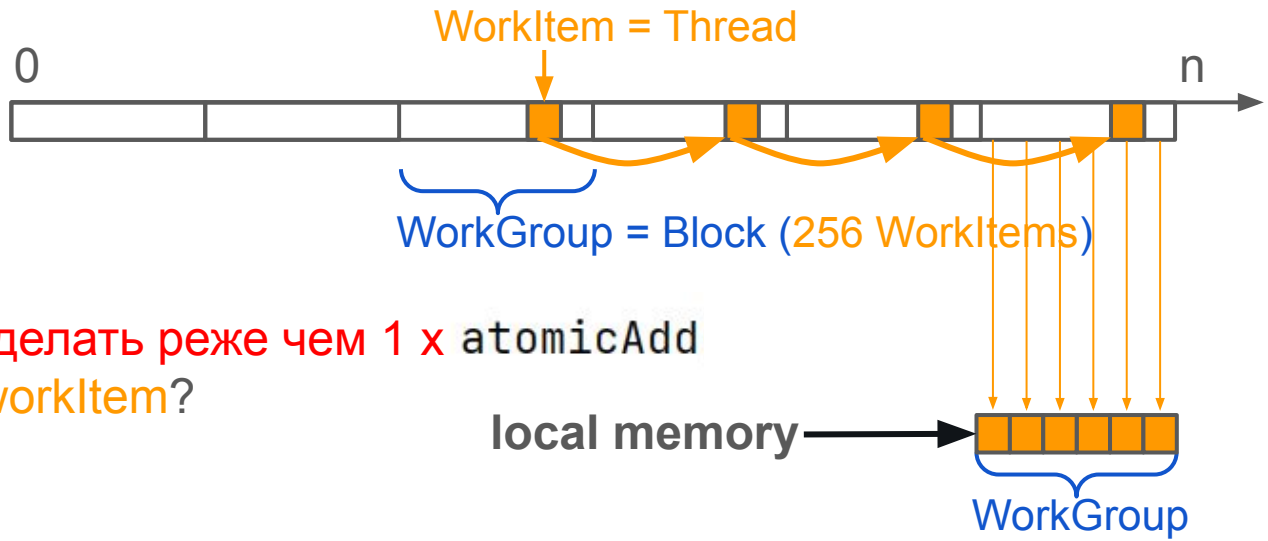


как делать реже чем 1 x `atomicAdd`
на workItem?

05: GPU - один **atomicAdd** на **WorkGroup**

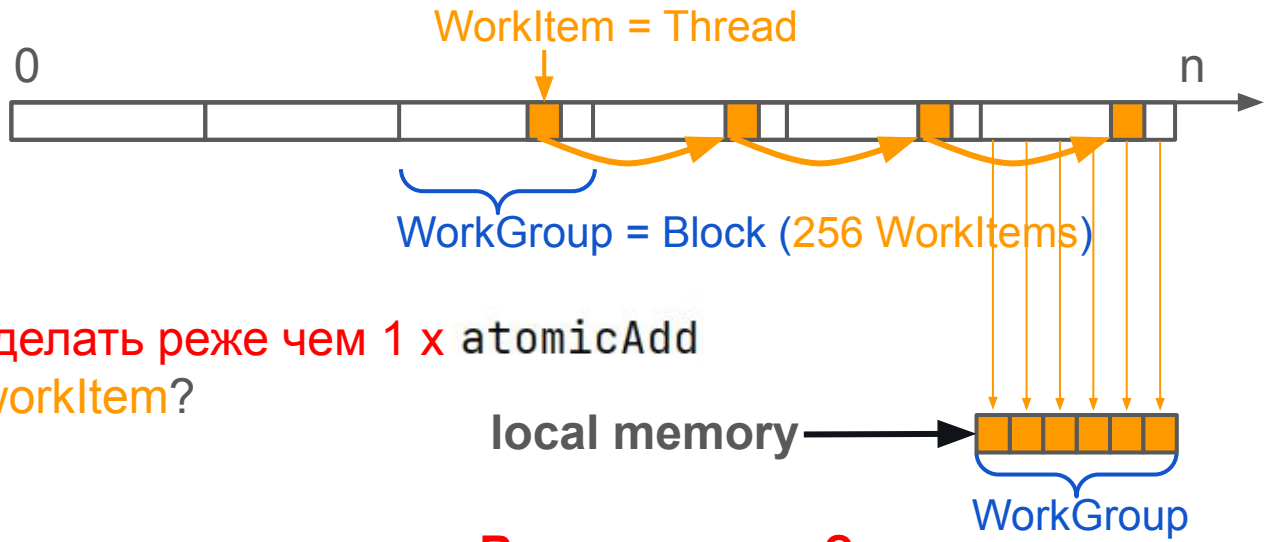


05: GPU - один **atomicAdd** на **WorkGroup**

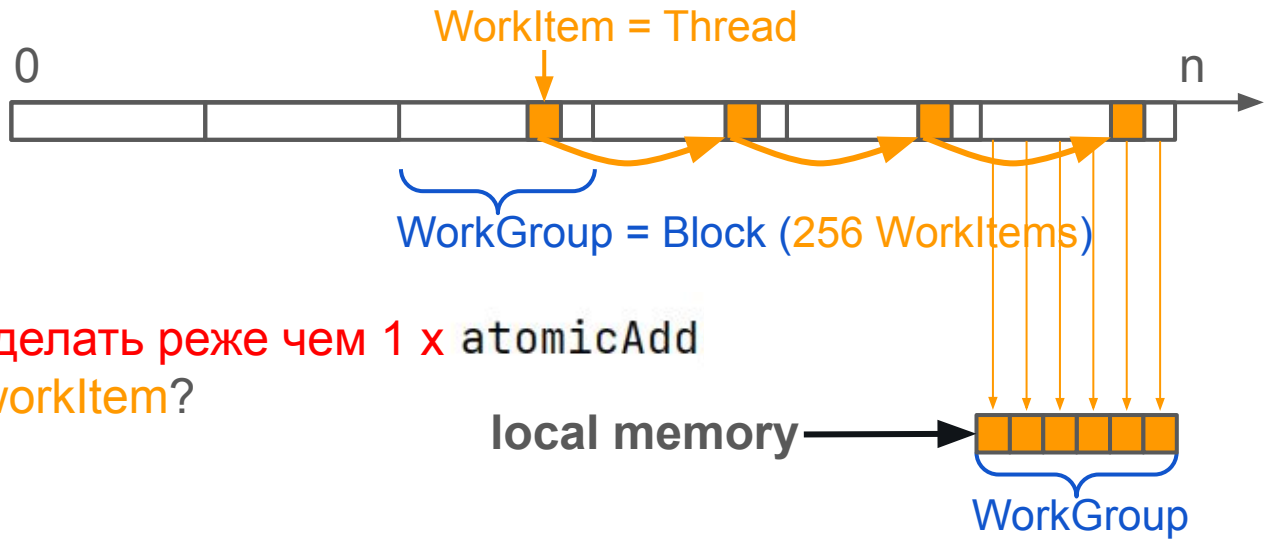


WorkItem=0 (**master**-thread): sum

05: GPU - один **atomicAdd** на **WorkGroup**



05: GPU - один **atomicAdd** на **WorkGroup**

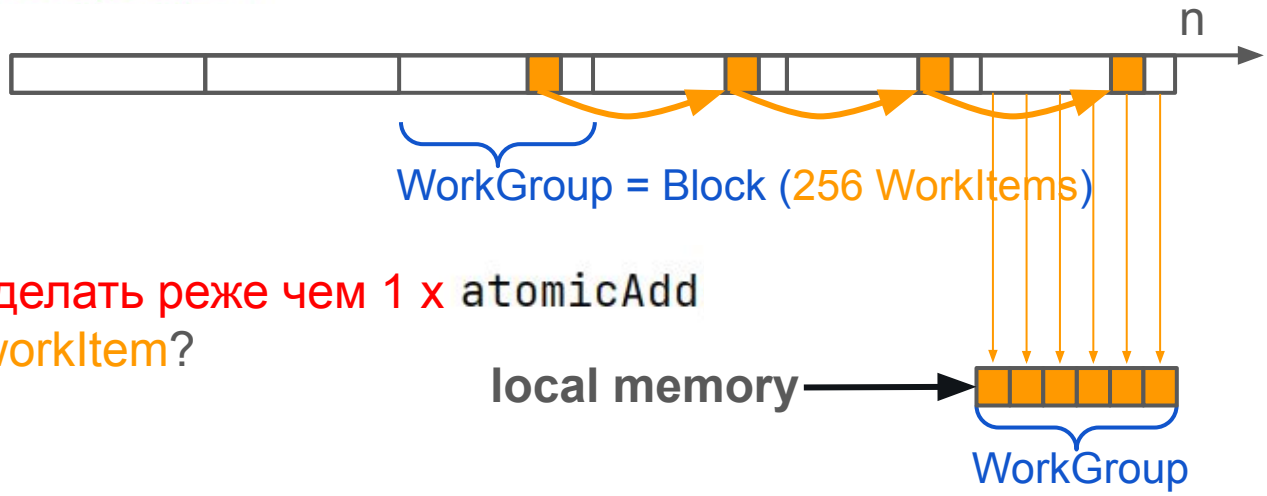



```
barrier(CLK_LOCAL_MEM_FENCE);
```

WorkItem=0 (**master**-thread):

```
if (index < n) {
    workgroup_data[get_local_id(0)] = a[index];
} else {
    workgroup_data[get_local_id(0)] = 0;
}
barrier(CLK_LOCAL_MEM_FENCE);
```

Запись нейтрального элемента (**padding**)

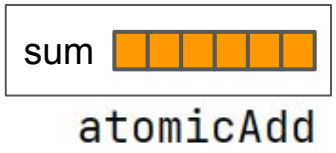


как делать реже чем 1 x `atomicAdd` на workItem?

```
barrier(CLK_LOCAL_MEM_FENCE);
```

А зачем padding?

WorkItem=0 (**master**-thread):

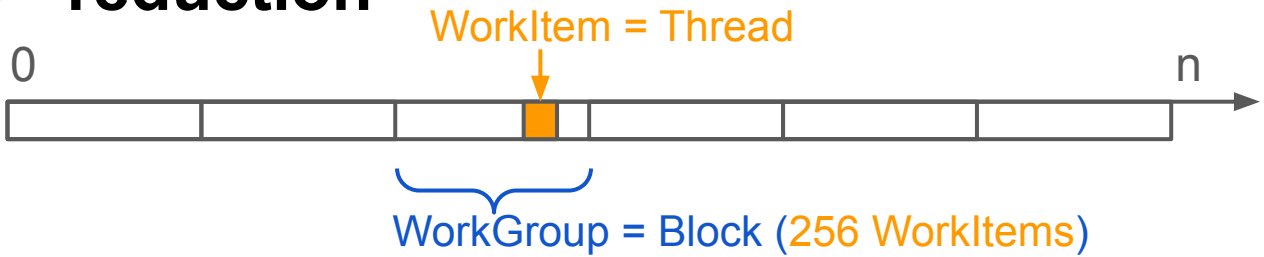


06: GPU - **идеи?**

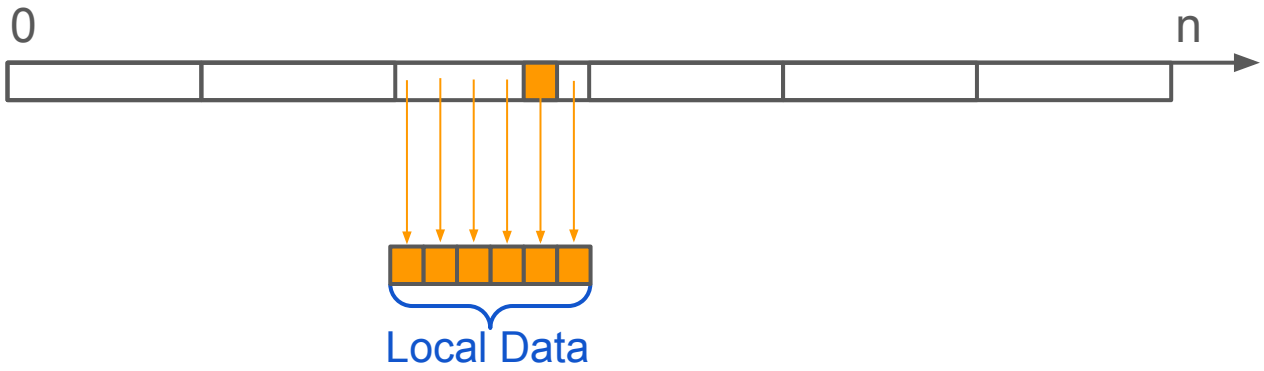
06: GPU - **идеи?**

как вообще без **atomicAdd?**

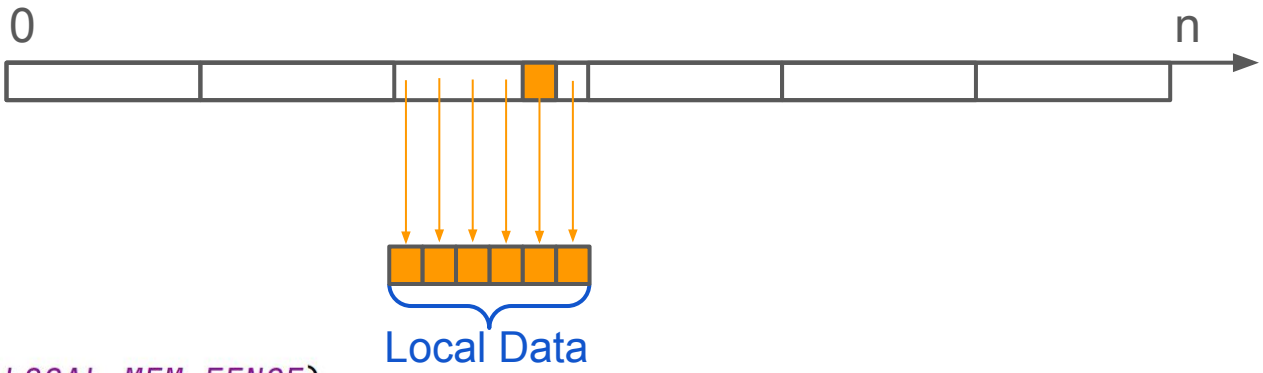
06: GPU - **reduction**



06: GPU - **reduction**

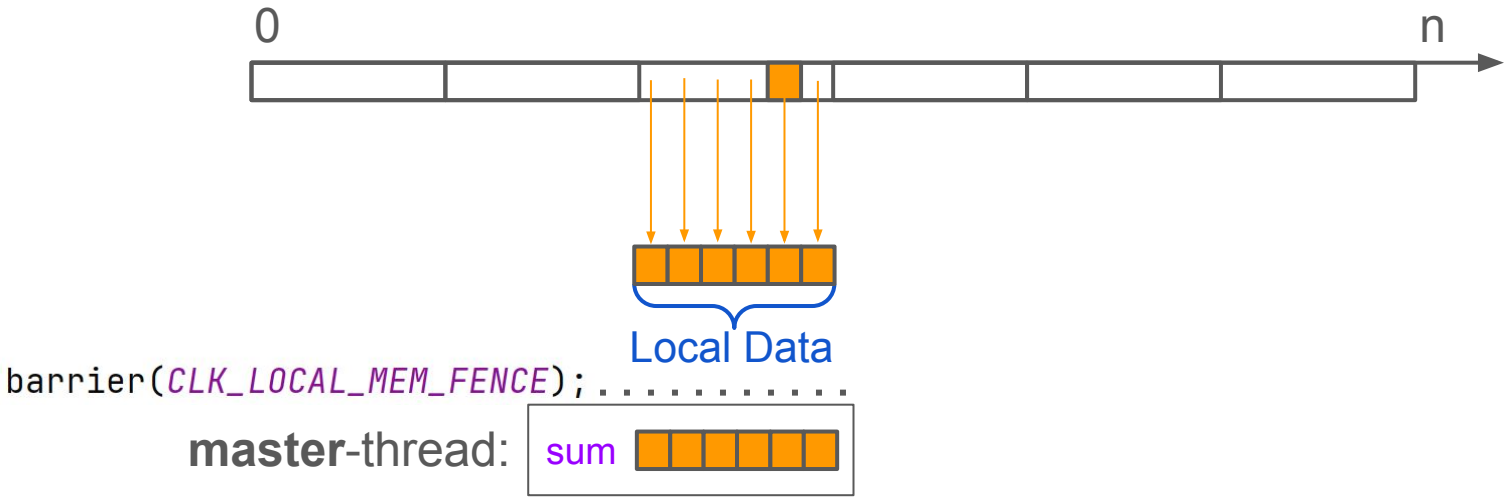


06: GPU - **reduction**

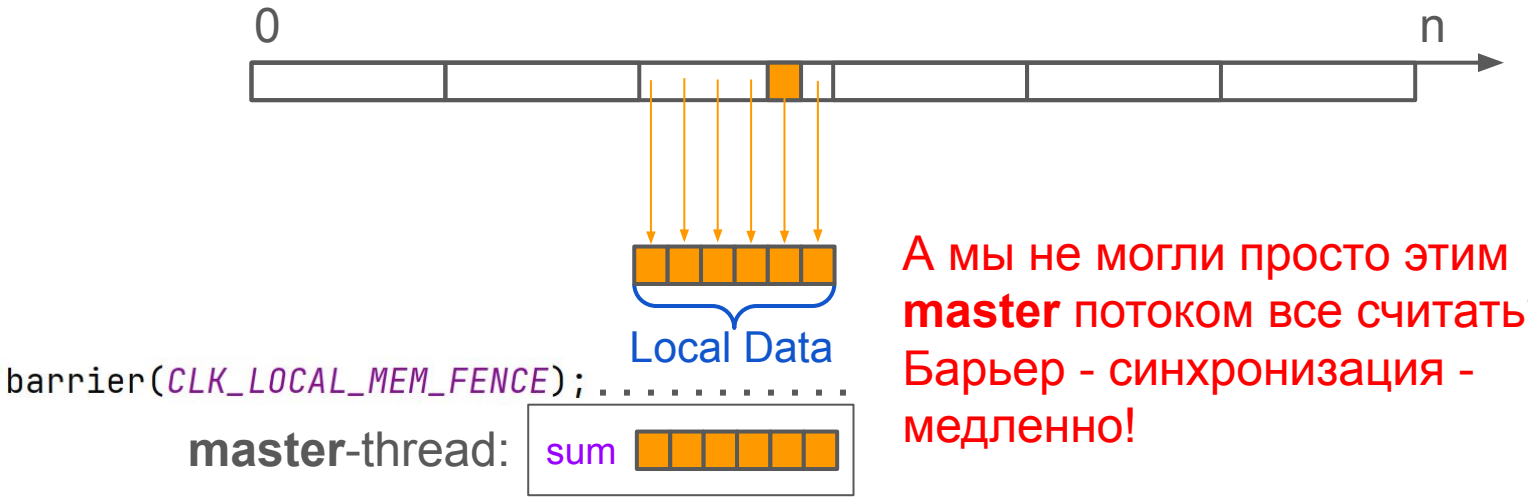



```
barrier(CLK_LOCAL_MEM_FENCE);.....
```

06: GPU - **reduction**

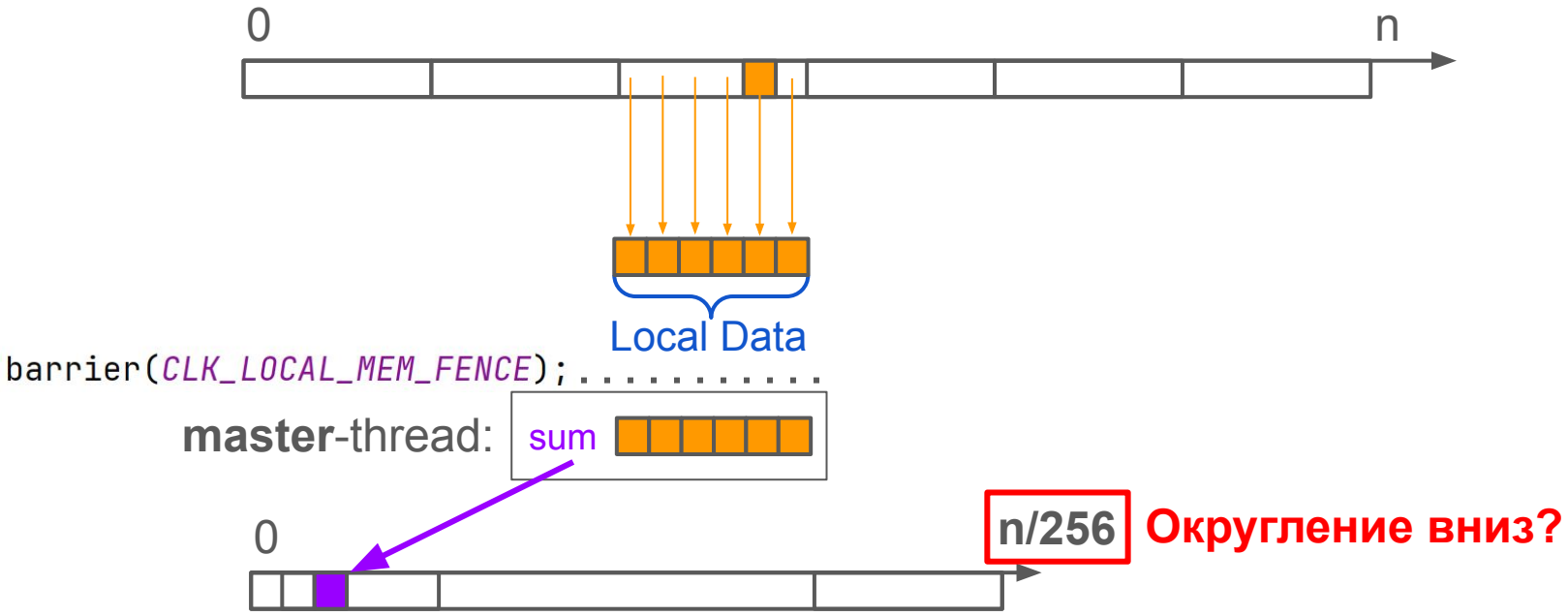


06: GPU - **reduction**

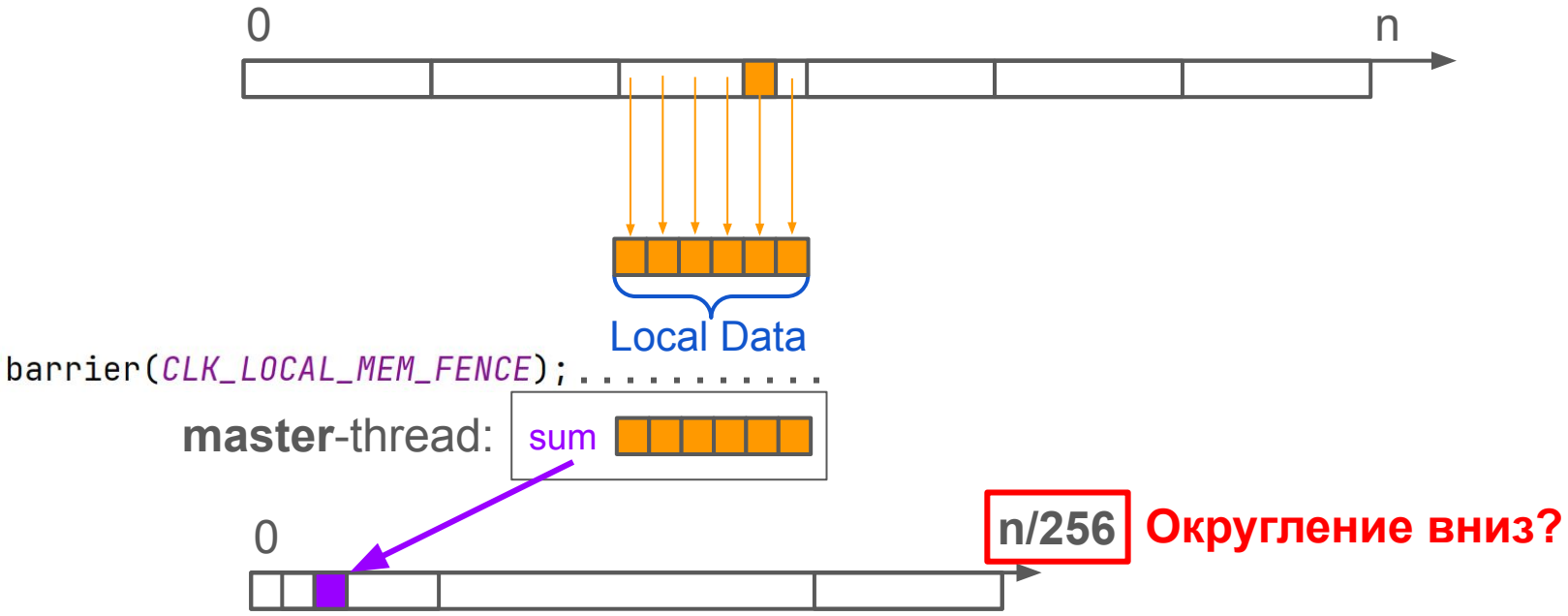


А мы не могли просто этим **master** потоком все считать?
Барьер - синхронизация - медленно!

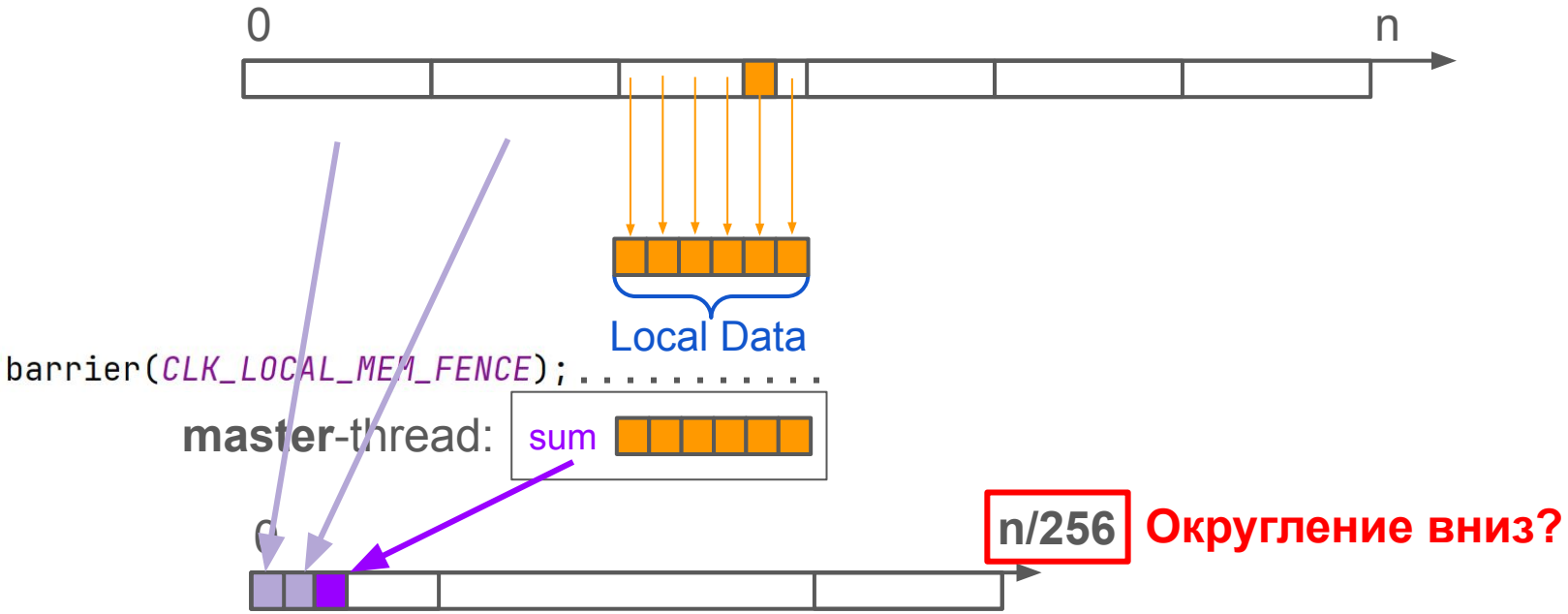
06: GPU - **reduction**



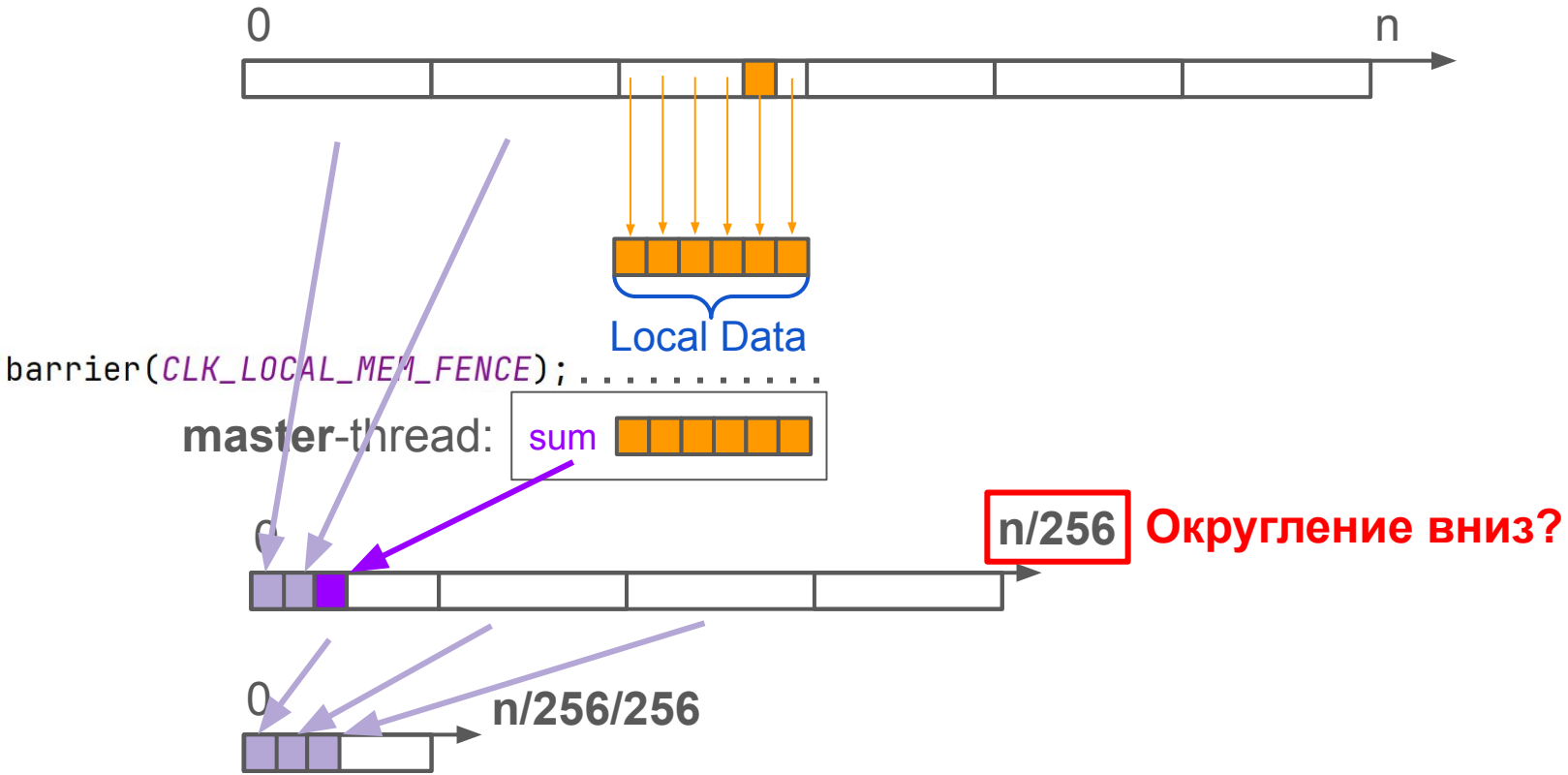
06: GPU - **reduction**



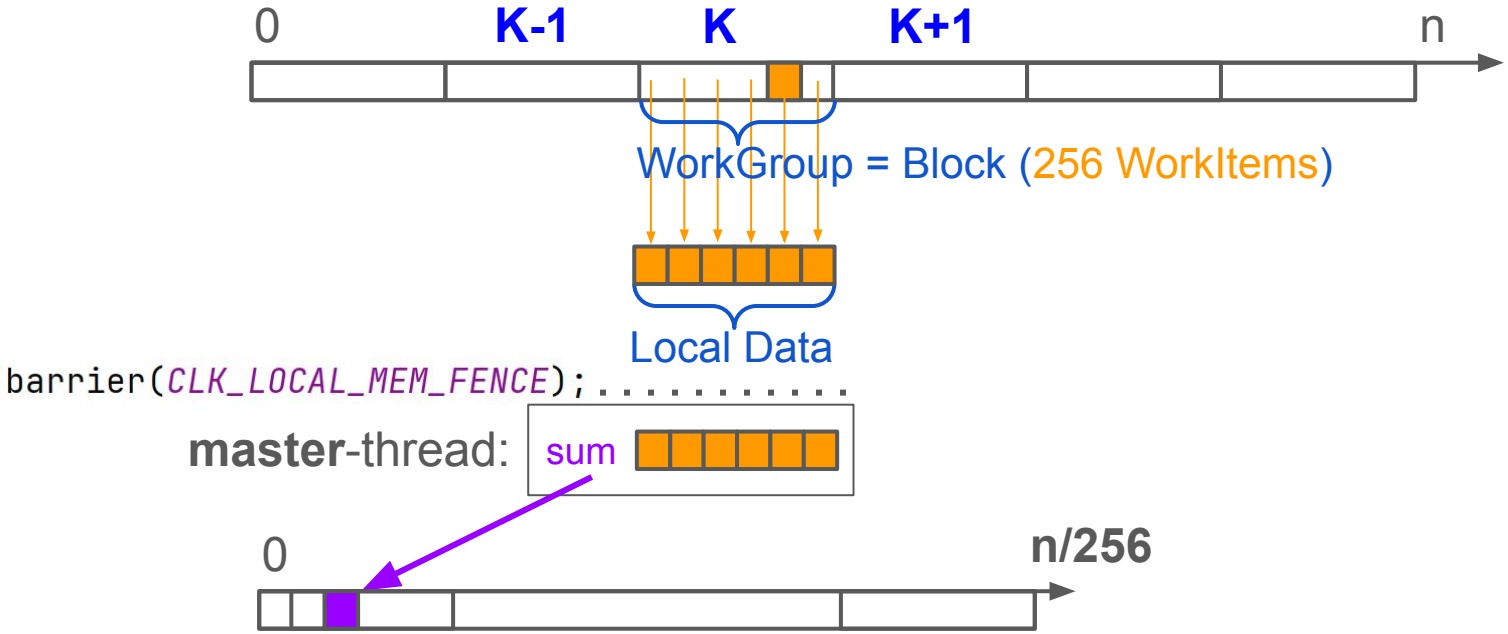
06: GPU - **reduction**



06: GPU - **reduction**

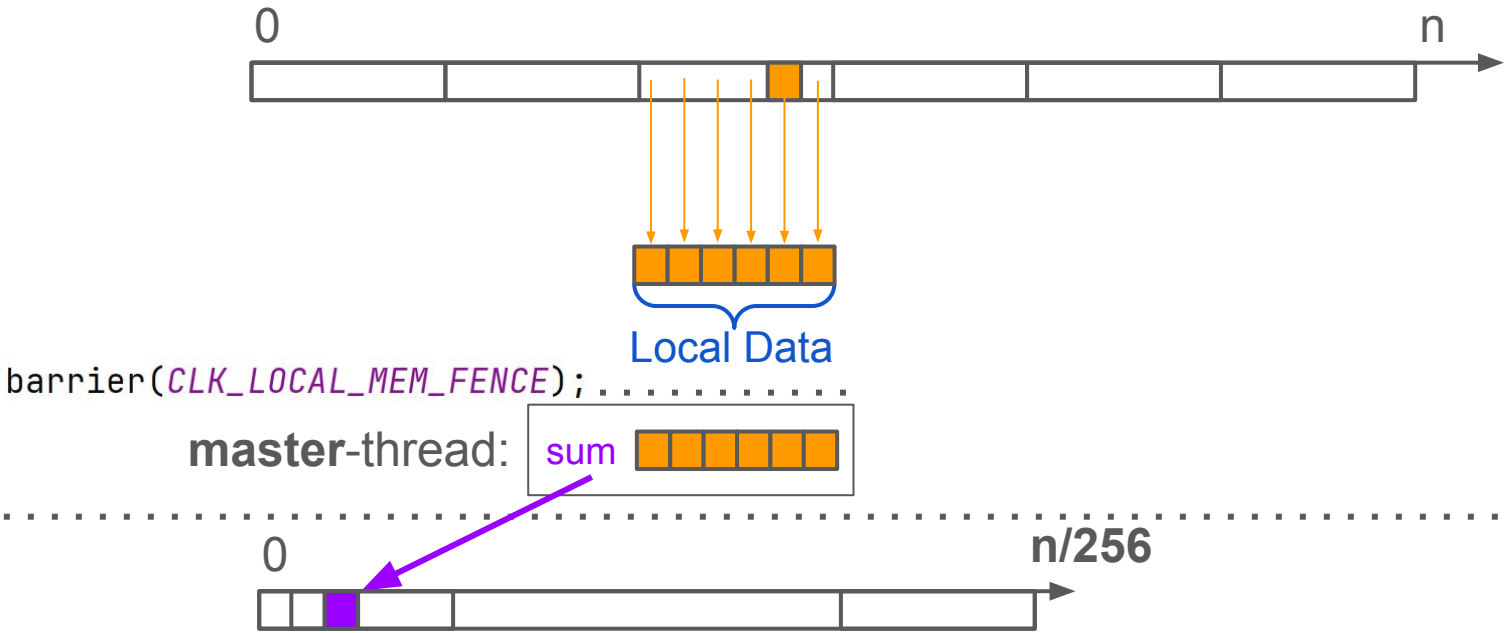


06: GPU - **reduction**



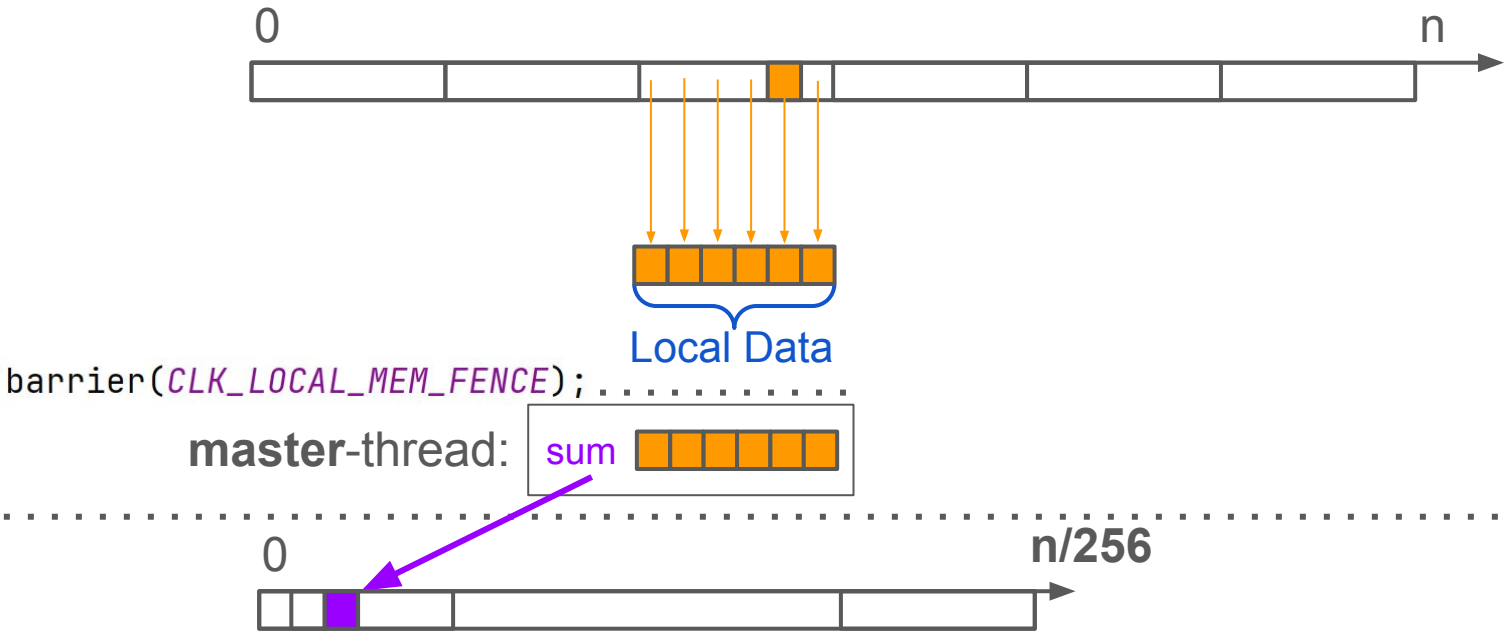
По какому индексу пишет **master**-thread если номер **workGroup = K**?

06: GPU - **reduction**



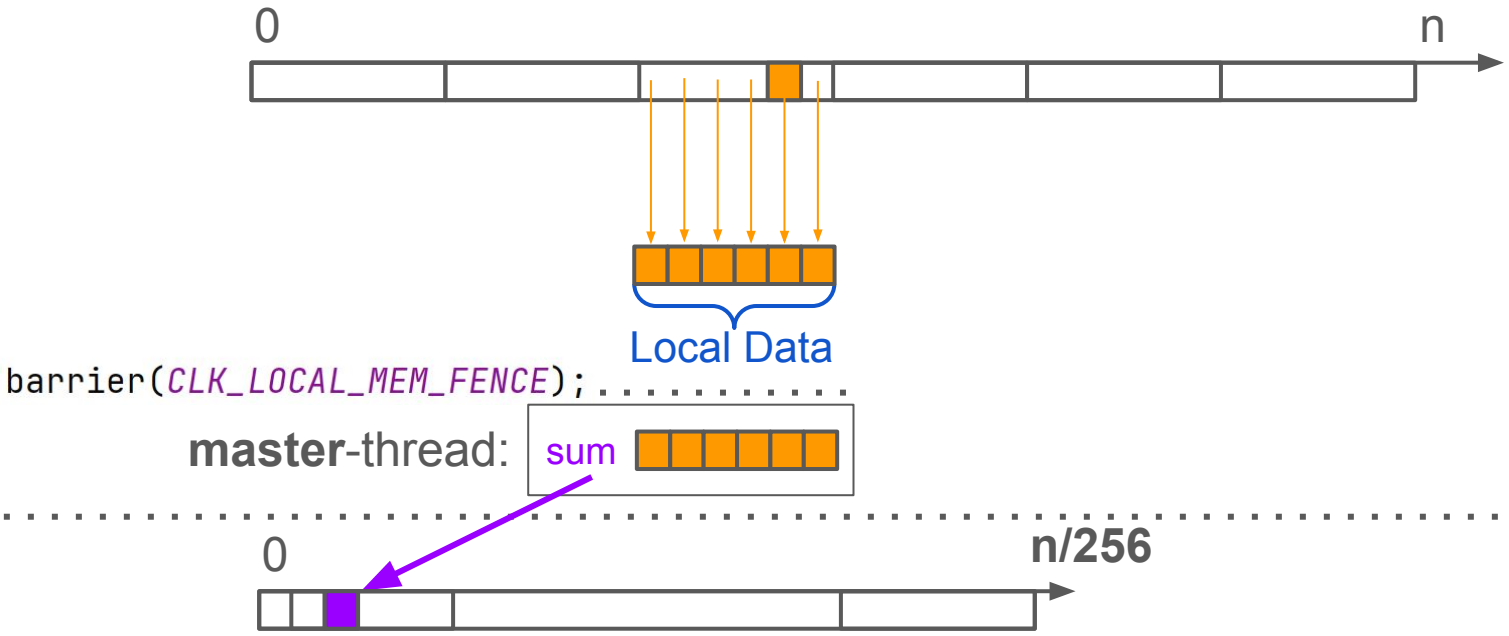
А как нам перейти на следующий шаг редукции?
Как **ГЛОБАЛЬНЫМ барьером** синхронизировать?

06: GPU - **reduction**



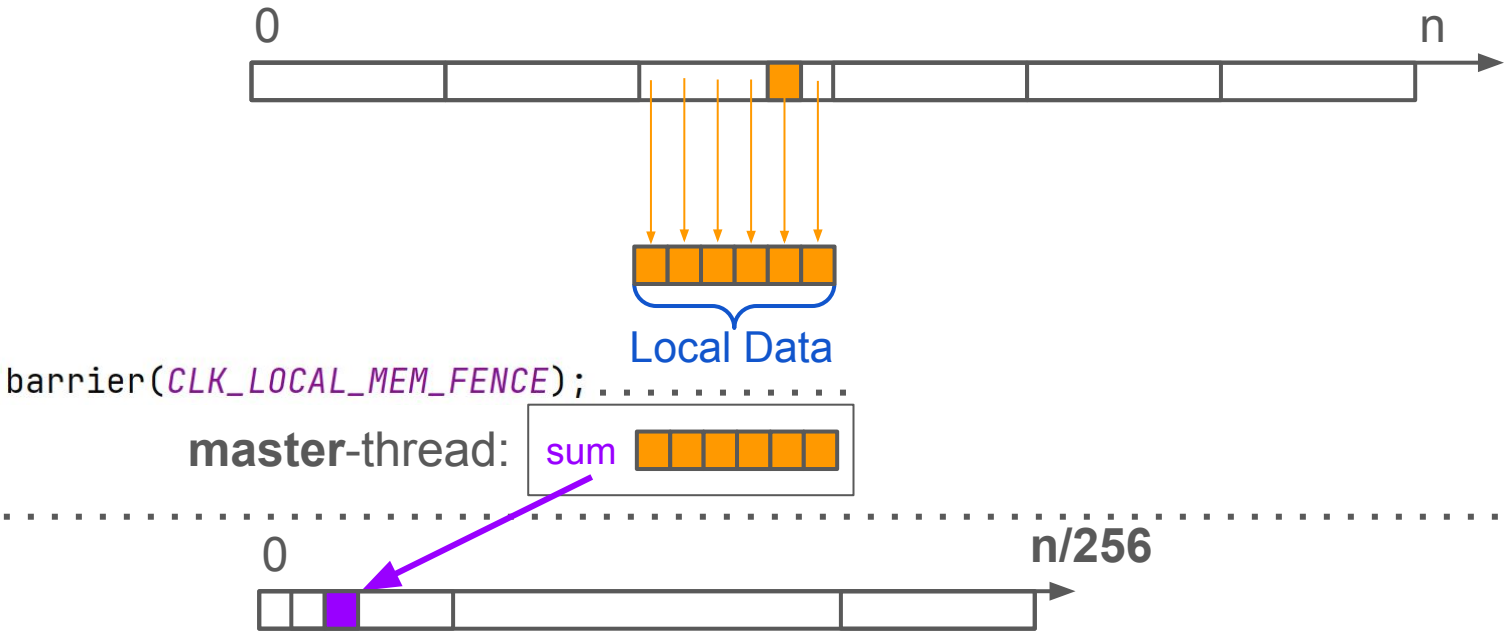
Сколько итого запусков **kernel**-ов?

06: GPU - **reduction**



А какая разница с **atomicAdd**?

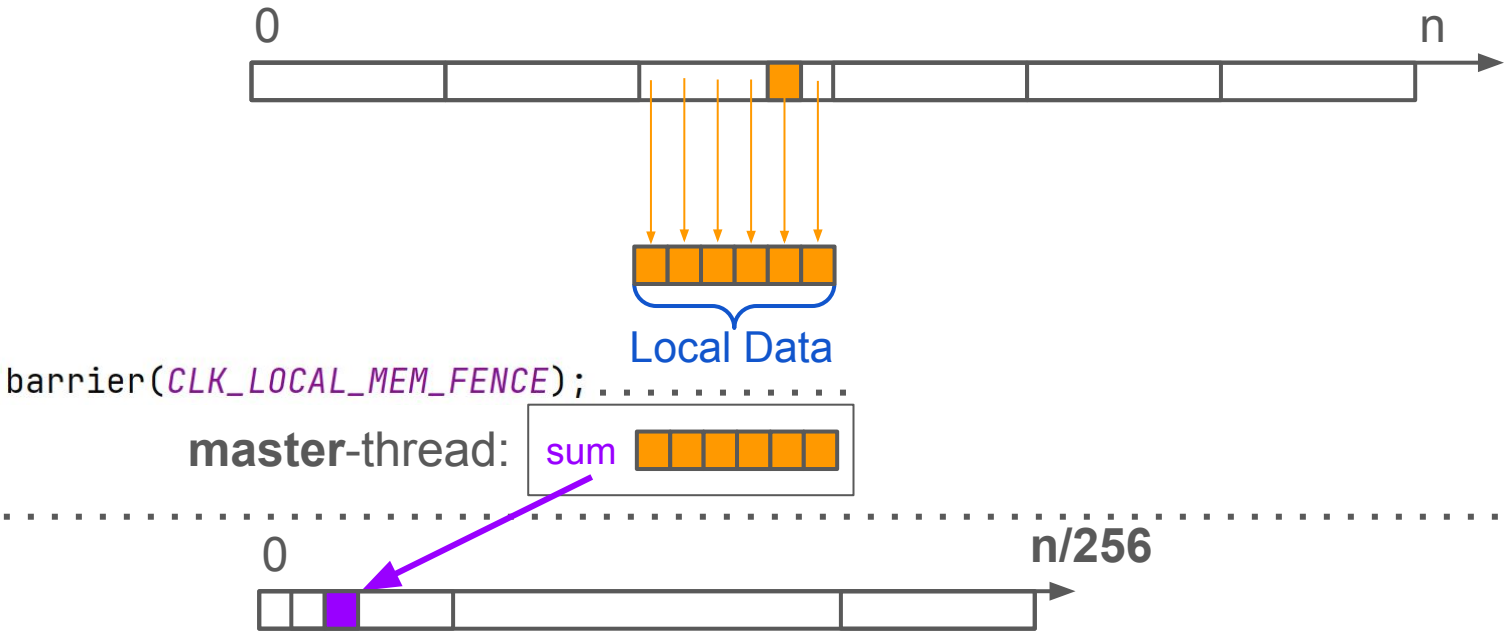
06: GPU - **reduction**



А какая разница с **atomicAdd**?

- детерминизм

06: GPU - **reduction**

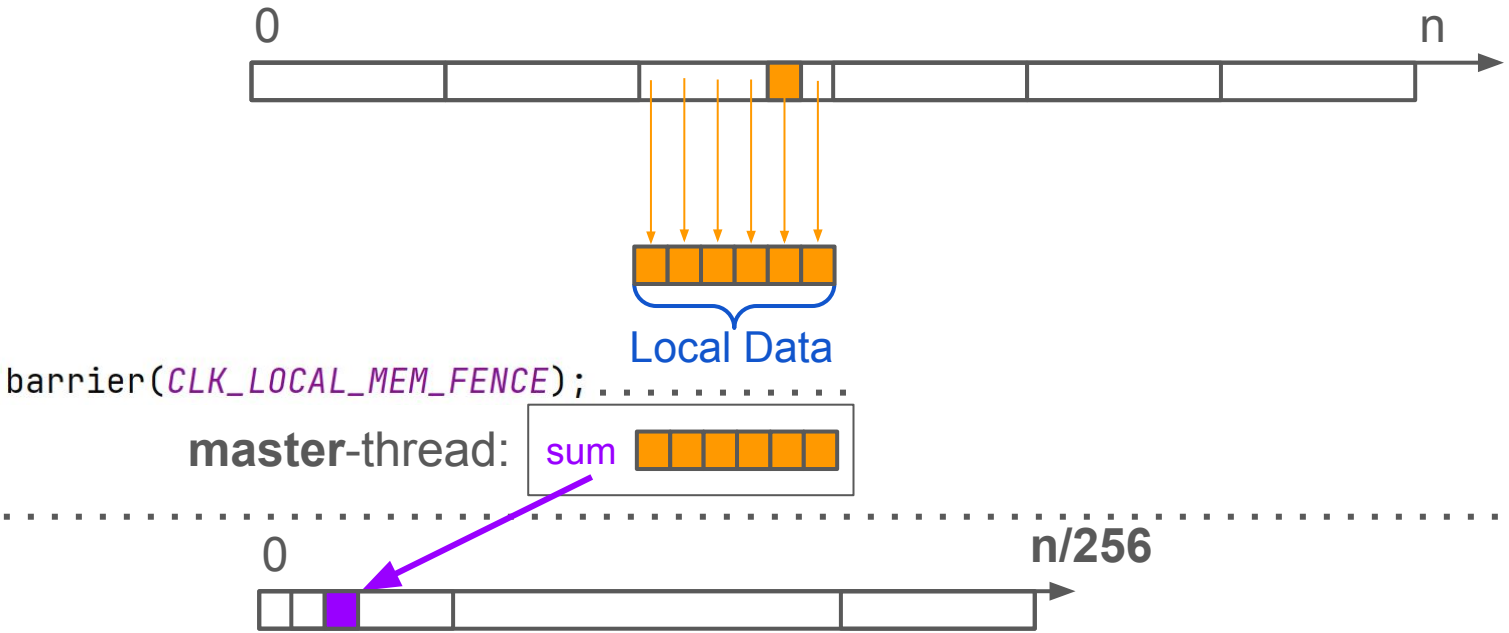


А какая разница с **atomicAdd**?

- детерминизм
- не все выразить через atomicAdd (arg-max минимум CAS = Compare And Set)

А как еще можно СИЛЬНО ускорить? (мы жестоко memory-bound)

06: GPU - **reduction**

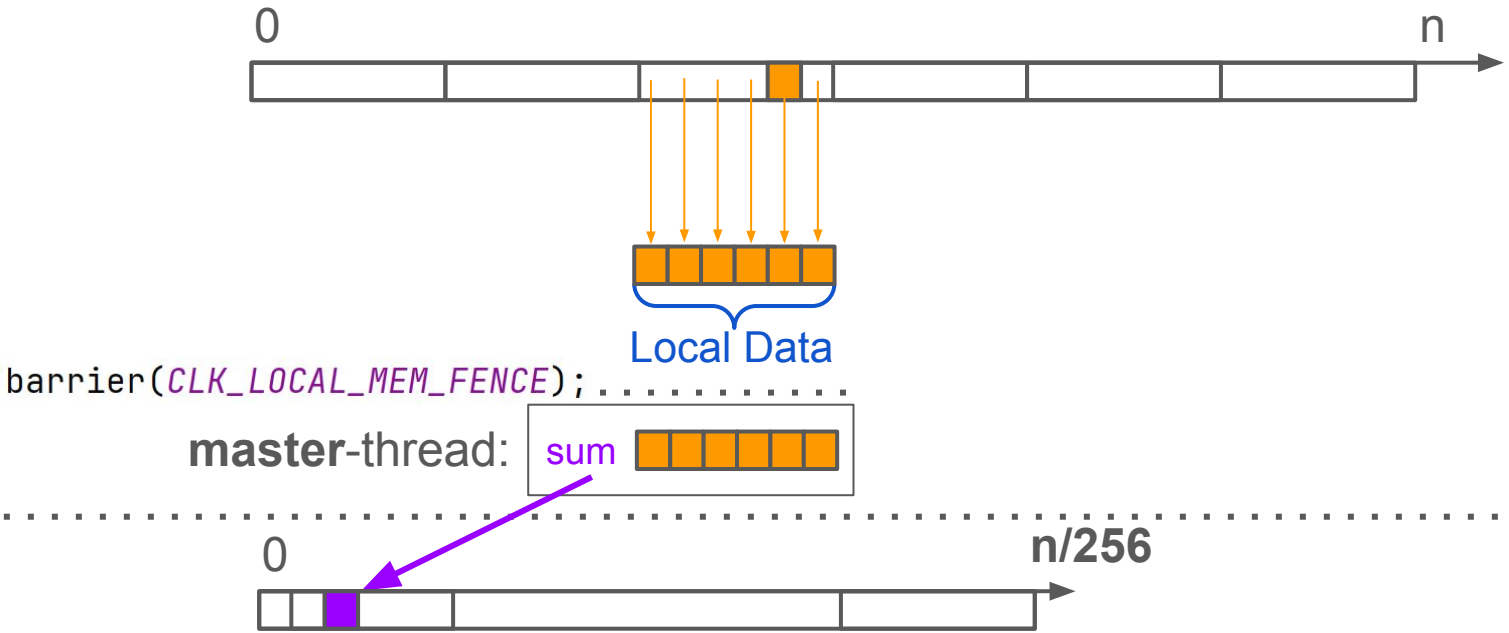


А какая разница с **atomicAdd**?

- детерминизм
- не все выразить через atomicAdd (arg-max минимум CAS = Compare And Set)

А как еще можно СИЛЬНО ускорить? Компрессия/поквантовка данных!

06: GPU - **reduction**



А какая разница с **atomicAdd**?

- детерминизм
- не все выразить через atomicAdd (arg-max минимум CAS = Compare And Set)

Глава 5: новое задание

Теоретическое. Фрактал Мандельброта. Сумма в массиве.

Теоретическое задание

1) Пусть на вход дан сигнал x[n], а на выход нужно дать два сигнала y1[n] и y2[n]:

```
y1[n] = x[n - 1] + x[n] + x[n + 1]
y2[n] = y2[n - 2] + y2[n - 1] + x[n]
```

Какой из двух сигналов будет проще и быстрее реализовать в модели массового параллелизма на GPU и почему?

Теоретическое задание

2) Предположим что размер warp/wavefront равен 32 и рабочая группа делится на warp/wavefront-ы таким образом что внутри warp/wavefront номер WorkItem по оси x меняется чаще всего, затем по оси y и затем по оси z.

Напоминание: инструкция исполняется (пусть и отмаскированно) в каждом потоке warp/wavefront если хотя бы один поток выполняет эту инструкцию неотмаскированно. Если не все потоки выполняют эту инструкцию неотмаскированно - происходит т.н. code divergence.

Пусть размер рабочей группы (32, 32, 1)

```
int idx = get_local_id(1) + get_local_size(1) * get_local_id(0);
if (idx % 32 < 16)
    foo();
else
    bar();
```

Произойдет ли code divergence? Почему?

Теоретическое задание

3) Как и в прошлом задании предположим что размер warp/wavefront равен 32 и рабочая группа делится на warp/wavefront-ы таким образом что внутри warp/wavefront номер WorkItem по оси x меняется чаще всего, затем по оси y и затем по оси z.

Пусть размер рабочей группы (32, 32, 1). Пусть data - указатель на массив float-данных в глобальной видеопамяти идеально выравненный (выравнен по 128 байтам, т.е. data % 128 == 0). И пусть размер кеш линии - 128 байт.

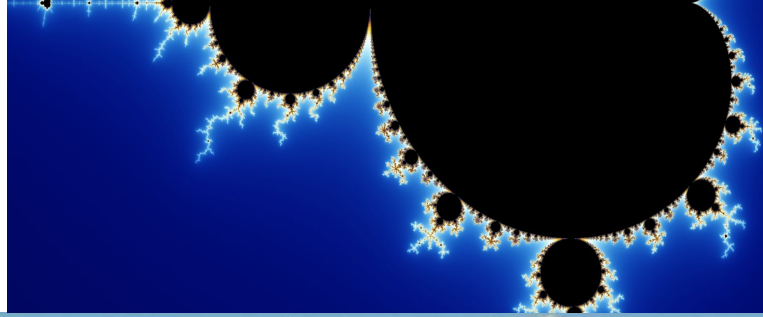
(a)
```
data[get_local_id(0) + get_local_size(0) * get_local_id(1)] = 1.0f;
```

(b)
```
data[get_local_id(1) + get_local_size(1) * get_local_id(0)] = 1.0f;
```

(c)
```
data[1 + get_local_id(0) + get_local_size(0) * get_local_id(1)] = 1.0f;
```

Будет ли данное обращение к памяти coalesced? Сколько кеш линий записей произойдет в одной рабочей группе?

Задание - Фрактал Мандельброта



AMD 7900XT

VS

AMD 9700XT

Как сравнить?

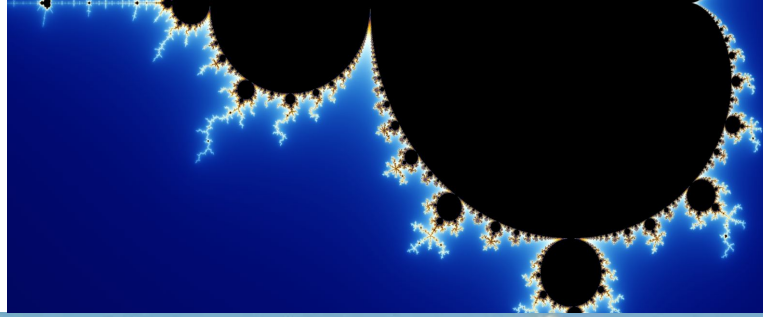
Как выбрать для задачи?

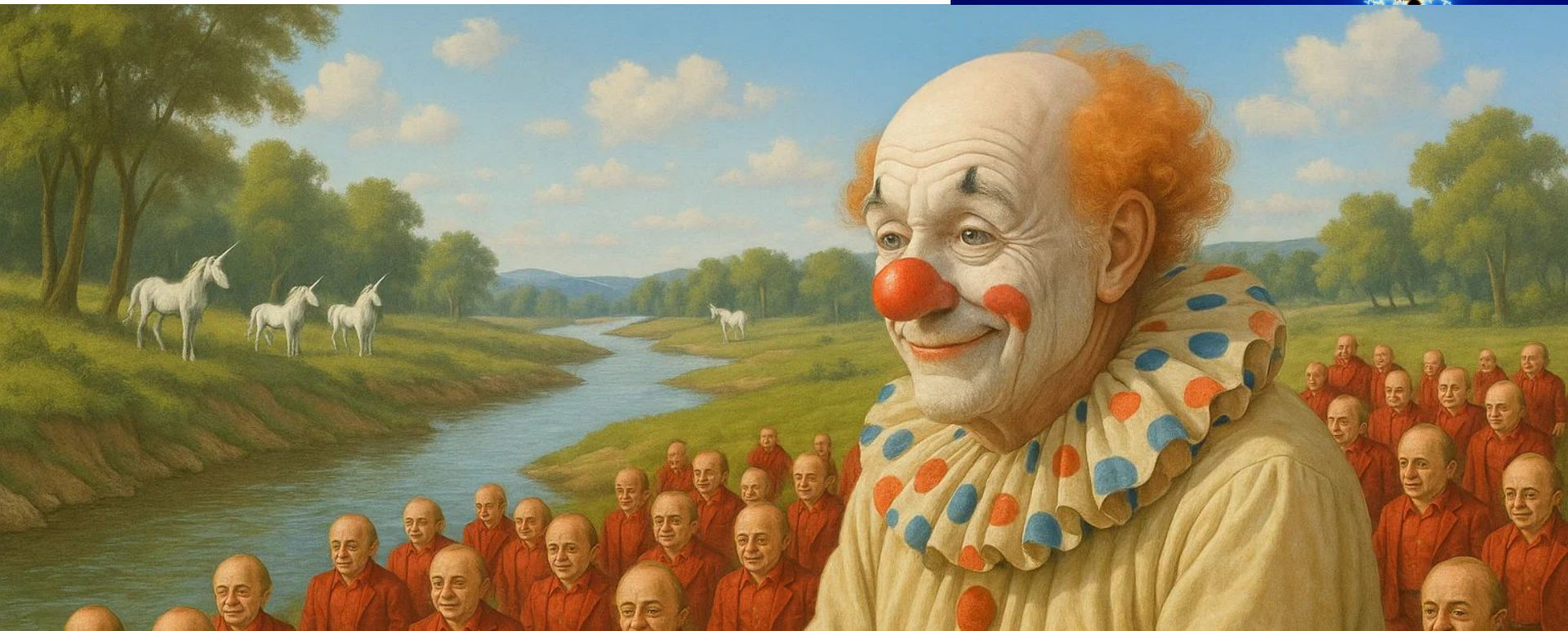
Задание - Фрактал Мандельброта

AMD 7900XT - FP32 **51.48** TFlops - **800.0** GB/s

AMD 9700XT - FP32 **48.66** TFlops - **644.6** GB/s

VS



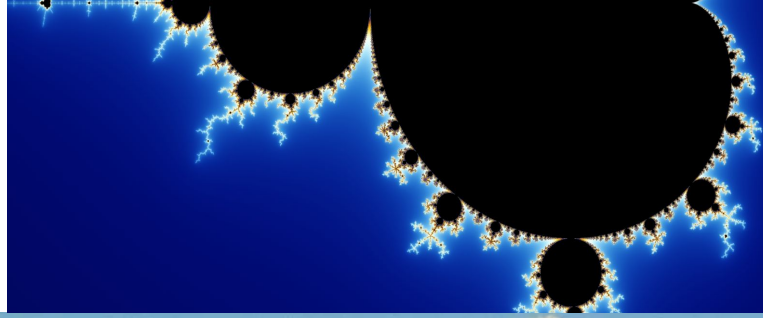


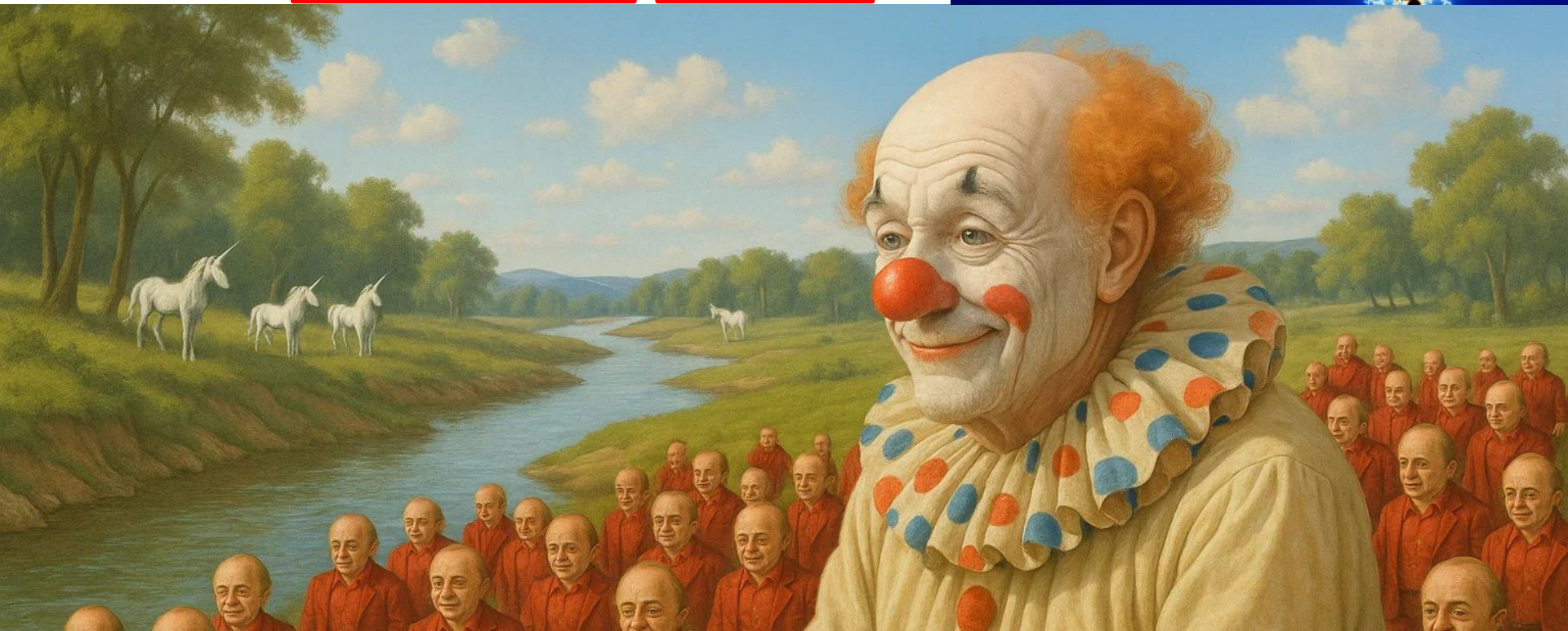
Задание - Фрактал Мандельброта

AMD 7900XT - FP32 **51.48** TFlops - **800.0** GB/s

AMD 9700XT - FP32 **48.66** TFlops - **644.6** GB/s

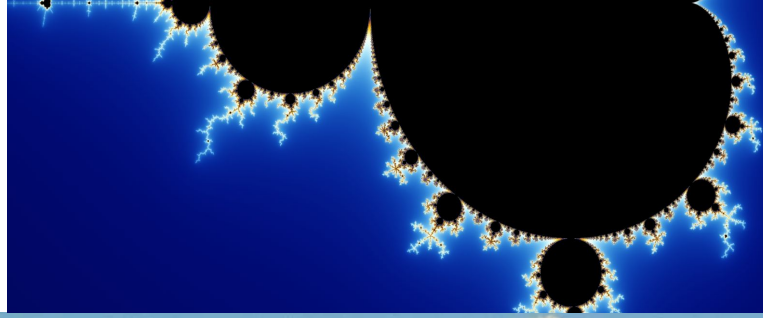
VS

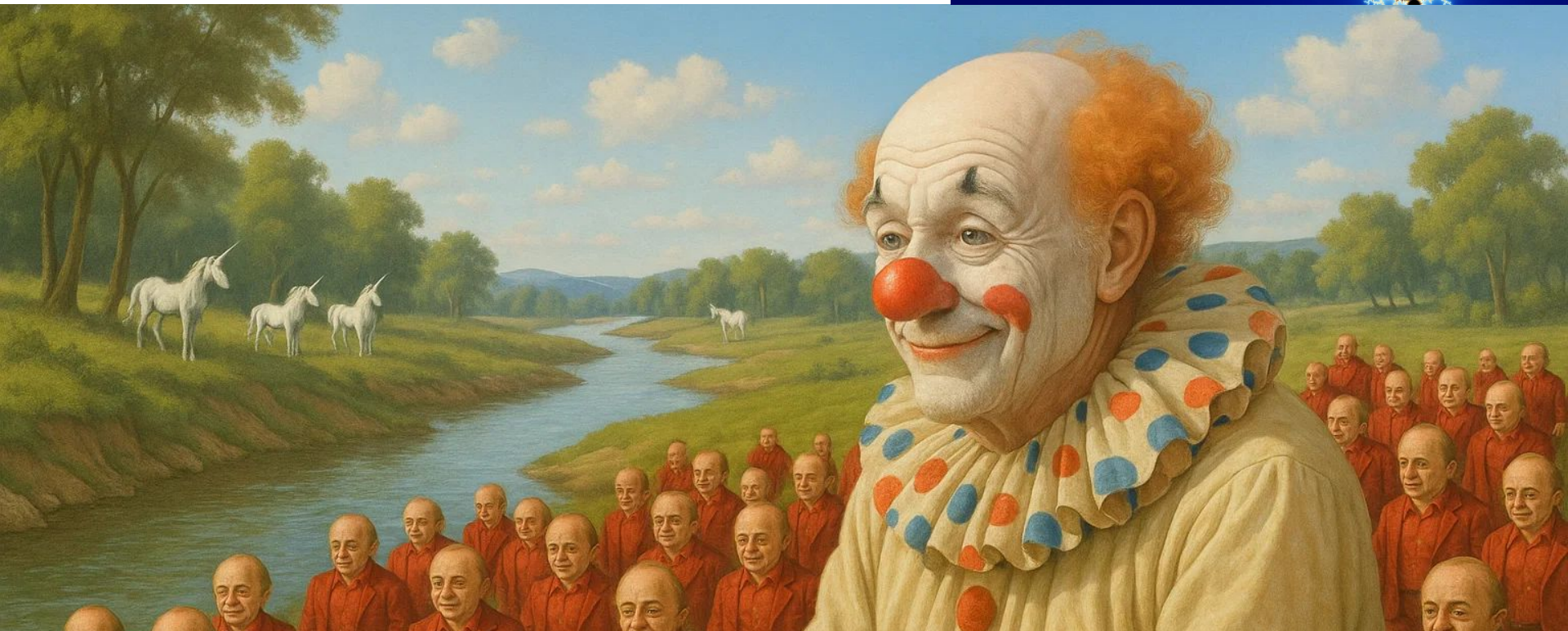




Задание - Фрактал Мандельброта

Давайте явно посчитаем сколько
GFlops получится достичь!





Задание - Фрактал Мандельброта

Давайте явно посчитаем сколько GFlops получится достичь!

Задание - Суммирование чисел в массиве

Попробуем глобальную редукцию!

Советы по отладке суммирования массива

1) Автоматика:
   - **CUDA** - **Compute Sanitizer** (бывший cuda-memcheck)
   - **OpenCL** - **Oclgrind**

Советы по отладке суммирования массива

1) Автоматика:
   - **CUDA** - **Compute Sanitizer** (бывший cuda-memcheck)
   - **OpenCL** - **Oclgrind**
2) Упрощайте + смотрите что происходит:
   - сделайте маленький **n=16**

Советы по отладке суммирования массива

1) Автоматика:
   - **CUDA** - **Compute Sanitizer** (бывший cuda-memcheck)
   - **OpenCL** - **Oclgrind**
2) Упрощайте + смотрите что происходит:
   - сделайте маленький **n=16**
   - **std::cout << “values=” << stats::vectorToString(values_gpu.readVector()) << std::endl;**

Советы по отладке суммирования массива

1) Автоматика:
   - **CUDA** - **Compute Sanitizer** (бывший cuda-memcheck)
   - **OpenCL** - **Oclgrind**
2) Упрощайте + смотрите что происходит:
   - сделайте маленький **n=16**
   - **std::cout << “values=” << stats::vectorToString(values_gpu.readVector()) << std::endl;**
   - **printf(“index=%d value=%d\n”, index, value);** (в Vulkan: **debugPrintfEXT**(...))

Советы по отладке суммирования массива

1) Автоматика:
   - **CUDA** - **Compute Sanitizer** (бывший cuda-memcheck)
   - **OpenCL** - **Oclgrind**
2) Упрощайте + смотрите что происходит:
   - сделайте маленький **n=16**
   - **std::cout << “values=” << stats::vectorToString(values_gpu.readVector()) << std::endl;**
   - **printf(“index=%d value=%d\n”, index, value);** (в Vulkan: **debugPrintfEXT**(...))
   - **rassert(<condition>, 435245341);**

Советы по отладке суммирования массива

1) Автоматика:
   - **CUDA** - **Compute Sanitizer** (бывший cuda-memcheck)
   - **OpenCL** - **Oclgrind**
2) Упрощайте + смотрите что происходит:
   - сделайте маленький **n=16**
   - **std::cout << “values=” << stats::vectorToString(values_gpu.readVector()) << std::endl;**
   - **printf(“index=%d value=%d\n”, index, value);** (в Vulkan: **debugPrintfEXT**(...))
   - **rassert(<condition>, 435245341);**
3) **Как проанализировать насколько мы близки к предельной скорости суммирования?**

Советы по отладке суммирования массива

1) Автоматика:
   - **CUDA** - **Compute Sanitizer** (бывший cuda-memcheck)
   - **OpenCL** - **Oclgrind**
2) Упрощайте + смотрите что происходит:
   - сделайте маленький **n=16**
   - **std::cout << “values=” << stats::vectorToString(values_gpu.readVector()) << std::endl;**
   - **printf(“index=%d value=%d\n”, index, value);** (в Vulkan: **debugPrintfEXT**(...))
   - **rassert(<condition>, 435245341);**
3) **Как проанализировать насколько мы близки к предельной скорости суммирования? Speed-of-Light анализ!**


CS Space
Клуб технологий и науки
Вопросы?
@UnicornGlade
@PolarNick239
polarnick239@gmail.com
Николай Полярный